UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Estudios de Internet: Cálculo de conexiones Contenidas

Javier Santos Alonso

Tutor: Javier Aracil Rico

Julio 2015

Estudios de Internet: Cálculo de Conexiones Contenidas

AUTOR: Javier Santos Alonso

TUTOR: Javier Aracil Rico

Dpto. TEC

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Julio de 2015

Resumen (castellano)

En este Trabajo de Fin de Grado se busca mostrar un algoritmo para calcular las conexiones contenidas, con su respectivo análisis teórico y una corroboración práctica realizada.

Una conexión está contenida en otra, si y solo si, la conexión contenedora y contenida tienen la misma IP origen y destino, así como los mismos puertos, y se cumple también que el tiempo de inicio de la conexión contenedora es menor que el tiempo de inicio de la contenida, y el tiempo de fin de la conexión contenedora es mayor que el tiempo de fin de la conexión contenida.

Para realizar este cálculo de conexiones contenidas se han realizado varias iteraciones, la primera de todas fue el diseño de un algoritmo de fuerza bruta, para saber cuál era la cota máxima que podíamos tener, este algoritmo de fuerza bruta tenía un coste teórico de $O(N^2)$.

Tras la realización de este algoritmo y ver que el tiempo de cálculo era de 48 horas aproximadamente, se pasó primero a eliminar las conexiones que ya habían pasado en nuestra línea temporal, y en consiguiente no podían tener más conexiones contenidas, porque no cumplían la condición de tiempo de fin, ya que el tiempo de inicio de la conexión en la que nos encontrábamos no podía ser menor que el tiempo de fin de la posible contenedora. Esto realizó una reducción en la constante del orden, pero seguía manteniéndose en un tiempo de ejecución muy elevado.

Después de esto, se eliminó las conexiones que tenían campos incompletos, es decir, se eliminaron las conexiones que no tenían un número de paquetes, ni un fin establecido o un inicio establecido así como las que tenían un consumo en bytes igual a 0. Este cambio volvió a reducir la constante global pero aún no era suficiente, ya que en tiempo de ejecución era muy costoso.

Tras esto se pasó a realizar una ordenación de los tiempos de inicio para que las conexiones contenidas estuviesen situadas en cascada y así poder reducir el orden de nuestro problema. Aun así aun no era suficiente, porque aunque si se borraban conexiones más rápido el coste de un algoritmo de ordenación de divide y vencerás es de orden $O(N*\log(N))$, lo que hizo que aumentará un poco el coste, pero si se eliminaban más rápido las conexiones.

Por último y dándonos cuenta de que se podía descartar la variable utilizada para las conexiones expiradas debido a que se podía empezar desde la i-esima conexión hasta que la línea temporal se acabase el orden se vio reducido a O(N*log(N)), ya que el número de conexiones que puede almacenar un ordenador a la vez es finito luego nuestro orden N^2 quedaba acotado por una constante W y el número de conexiones N.

Los resultados finales dieron lugar a que el coste del cálculo del número de conexiones contenidas pasase de 48 horas aproximadamente para 3GB a 6 minutos, dando lugar al final de este trabajo de fin de grado.

Palabras clave

Conexión contenida, algoritmo, orden del algoritmo, tiempo de ejecución, conexión contenedora.

Abstract (English)

This Bachelor Thesis we want to show an algorithm to calculated the content connection, to explain it we will do theoretical analysis and a practical analysis.

One connection is content in other if and only if the content connection and the connection which contains the content connection has the same IP and the same port, and if the content connection has a final time lower than the connection which contains the content connections, and a higher initial time than the other.

To realize this algorithm we have done different iteration, the first one was doing a Brute-force algorithm, this algorithm was done because we want a maximum level we can get in this problem, the theoretical level was $O(N^2)$

After doing this algorithm and look at the execution time was around 48 hours, we decide to eliminate the connection which has been passed in our time line to do the algorithm more efficient. We can do this change because the final time condition will not appear if the connection is on an initial time which is higher than the final time of the connection we suppose that is a content connection. This change reduce the constant of the level N^2 but the execution time was really long.

Later we eliminated the connection which has incomplete fields. It means the connection which does not have for example any bytes use. This reduction do a better execution time but was not enough.

Afterwards we do a sorting of the initial time. It was because we want a cascade of the connection to improve the speed of the algorithm. With this change we want to remove more connection with the variable of expired connection, but the order of divide and conquer algorithm was not enough good to improve the time as much as we want.

At the end if we remove the expired connection variable (with the sorting) and we start the iteration of the loop in the i-esim connection until the time is over, because of the number of content connection a machine can have we got a reduction in the algorithm to $O(N*\log(N))$.

The final result we obtain was 6 minutes for 3 GB that was enough for the Bachelor Thesis.

Keywords (inglés)

Content Connection, algorithm, algorithm order, and execution time.

Agradecimientos

Quiero agradecer a todas las personas que me han ayudado en la realización de este Trabajo de Fin de grado. En primer lugar, me gustaría agradecer a Javier Aracil la ayuda que me ha prestado como tutor y como miembro del profesorado de la Escuela Politécnica Superior, ya que sin el este trabajo no podría haber sido realizado. También me gustaría agradecer a HPCN la ayuda prestada, ya que ellos fueron los que me facilitaron los ficheros utilizados para el desarrollo de este trabajo.

Por otra parte también me gustaría agradecerle a Carlos Ramos que fue quien me ayudó con una de las iteraciones del algoritmo. Por último me gustaría también agradecer a todos mis compañeros de curso por su apoyo y ayuda a lo largo de la carrera, así como a mis padres que me han animado a seguir con esta carrera en los momentos más difíciles. Muchas gracias a todos.

INDICE DE CONTENIDOS

1 INTRODUCCIÓN 1

1.1 MOTIVACIÓN 1
1.2 OBJETIVO 2
1.3 CONTEXTO 2
1.4 ESTRUCTURA DEL DOCUMENTO 4

2 CÁLCULO TEÓRICO DEL COSTE DEL ALGORITMO 7

2.1 INTRODUCCIÓN 7
2.2 ALGORITMO DE FUERZA BRUTA 7
2.3 ALGORITMO DE FUERZA BRUTA CON ELIMINACIÓN DE CONEXIONES EXPIRADAS 8
2.4 ALGORITMO DE CONEXIONES EXPIRADAS CON ELIMINACIÓN DE CONEXIONES NO VALIDAS 11
2.5 ALGORITMO DE ELIMINACIÓN CON ORDENACIÓN DE CONEXIONES CON RESPECTO AL TIEMPO 13
2.6 ALGORITMO FINAL 16
2.7 CONCLUSIÓN 19

3 COSTE REAL DEL ALGORITMO 21

3.1 INTRODUCCIÓN 21
3.2 ALGORITMO DE FUERZA BRUTA 21
3.3 ALGORITMO DE FUERZA BRUTA CON ELIMINACIÓN DE CONEXIONES EXPIRADAS 23
3.4 ALGORITMO DE CONEXIONES EXPIRADAS CON ELIMINACIÓN DE CONEXIONES NO VALIDAS 26
3.5 ALGORITMO DE ELIMINACIÓN CON ORDENACIÓN 28
3.6 ALGORITMO FINAL 30
3.7 CONCLUSIÓN 33

4 CONCLUSIONES Y ESTUDIOS FUTUROS 35

4.1 CONCLUSIONES TEÓRICAS Y PRÁCTICAS 35
4.2 ESTUDIOS DE FUTURO Y POSIBLES MEJORAS 35
4.2.1 Cambio del algoritmo de ordenación *35*
4.2.2 Añadir multiproceso en el bucle del cálculo de las conexiones contenidas *36*

GLOSARIO DE TERMINOS 39

ANEXO 41

A CAMPOS DEL FICHERO 41
B EJEMPLO DE CONEXIONES QUE APARECEN EN EL FICHERO 46
C EJEMPLO DE SALIDA DEL PROGRAMA FINAL 47

INDICE DE FIGURAS

FIGURA 1-1.3: DEFINICIÓN DE CONEXIÓN CONTENIDA 3

FIGURA 2-1.3 DEFINICIÓN DE CONEXIÓN CONCURRENTE 3

FIGURA 3-2.2: ESTIMACIÓN TEÓRICA DEL ALGORITMO DE FUERZA BRUTA 8

FIGURA 4-2.3: TIEMPO TEÓRICO DEL ALGORITMO DE FUERZA BRUTA CON ELIMINACIÓN DE CONEXIONES 10

FIGURA 5-2.4: ESTIMACIÓN TEÓRICA DEL ALGORITMO DE CONEXIONES EXPIRADAS CON ELIMINACIÓN DE CONEXIONES NO VALIDAS 13

FIGURA 6-2.5: ESTADO ANTERIOR A QUICKSORT DE LAS CONEXIONES 15

FIGURA 7-2.5: ESTADO POSTERIOR A QUICKSORT DE LAS CONEXIONES 15

FIGURA 8-2.5: TIEMPO TEÓRICO UTILIZADO QUICKSORT 16

FIGURA 9-2.6: TIEMPO TEÓRICO DEL ALGORITMO FINAL 18

FIGURA 10-3.2: COSTE REAL DEL ALGORITMO DE FUERZA BRUTA 23

FIGURA 11-3.3: RESULTADO PRÁCTICO DEL ALGORITMO DE FUERZA BRUTA CON ELIMINACIÓN DE CONEXIONES EXPIRADAS 25

FIGURA 12-3.4: RESULTADO PRÁCTICO DEL ALGORITMO DE ELIMINACIÓN DE CONEXIONES CON ELIMINACIÓN DE NO VALIDAS 27

FIGURA 13-3.5: TIEMPO PRÁCTICO DEL ALGORITMO DE ELIMINACIÓN CON ORDENACIÓN 29

FIGURA 14-3.5: TIEMPO PRÁCTICO DEL ALGORITMO FINAL 31

FIGURA 15-3.6: TIEMPO PRÁCTICO AUMENTANDO CONSIDERABLEMENTE EL NÚMERO DE CONEXIONES Y PARADO EL PROGRAMA PARA HACERLO MÁS VISIBLE 32

FIGURA 16-3.7: TIEMPOS PRÁCTICOS EN CONJUNTO 33

FIGURA 17-3.: FIGURA DE TIEMPOS FINALES SIN EL ALGORITMO DE FUERZA BRUTA 34

INDICE DE TABLAS

Tabla 1: Campos utilizados para eliminar las conexiones no validas……11

1 Introducción

1.1 Motivación

En los tiempos en los que nos encontramos una de las principales fuentes de ocio y de información es Internet. Internet ahora mismo se utiliza para realizar todo tipo de actividades, ya sea ver un video en YouTube, realizar una transacción bancaria, incluso apostar por tu equipo favorito de cualquier deporte.

Debido al gran uso de Internet citado anteriormente se produce la necesidad de analizar cómo están distribuidas las cargas en las diversas redes que lo generan. Puede darse el caso que se tenga un gran volumen de conexiones sobre una misma máquina, que puede ser por dos motivos, que sea un servidor que da servicios, lo que hace que se tengan muchas conexiones contenidas a lo largo del día, o que se estén produciendo ataques de DDoS (distributed denial-of-service) o de DoS (denial-of-service). Estas tres opciones generan un gran volumen de conexiones contenidas, y concurrentes las cuales se quieren analizar, para realizar estudios topológicos sobre estas redes.

Por estos motivos anteriores, lo buscado con la realización de este Trabajo de Fin de Grado es desarrollar un algoritmo capaz de calcular las conexiones contenidas de una red, en un tiempo razonable.

Puede parecernos que no es necesario pero ahora mismo como Internet es muy concurrido es difícil ver que máquinas tienen más carga que otro, y es por esto que se introduce el concepto de conexión contenida.

En general las conexiones contenidas nos permiten realizar un dimensionado de la carga del servidor, es por eso que se requiere de este programa. Además si se está produciendo un ataque por DoS esto también aparece reflejado en las conexiones contenidas entre un terminal y el/los servidores que lo están sufriendo.

Un ejemplo claro que se puede observar de DoS sucedió hace relativamente poco, el 27 de Marzo de 2013, una compañía realizó un ataque de DoS para privar al resto de usuarios de los servicios de su competidora. Este caso es tan sonado debido a que debido al spam que se generó se vieron ralentizaciones generalizadas e incluso llegó a afectar al nodo central de Londres

Por lo tanto se busca un programa que calcule las conexiones contenidas de una red sobre los datos recibidos de un snifer. A la hora de realizar el estudio se hablará de conexiones contenidas, el cálculo de conexiones concurrentes se puede obtener relajando la condición sobre las conexiones contenidas, esto se explicará más adelante en este mismo capítulo en la sección 1.3.

1.2 Objetivo

El objetivo de este trabajo es mostrar un algoritmo capaz de calcular las conexiones contenidas, así como un pequeño programa que permite en un tiempo inferior a un día calcular las conexiones contenidas.

Otro objetivo de este trabajo es mostrar los pasos de elaboración de un algoritmo frente a un problema sobre el cuál no se tiene un algoritmo anterior para su cálculo, así como mostrar cuando se debe cambiar de lenguaje para optimizar la velocidad de procesamiento.

1.3 Contexto

Este trabajo no contiene estado del arte porque sobre esta materia no se ha publicado nada hasta este momento, por lo que se utilizará esta sección para explicar el problema. Para tener una idea de cómo comienza el desarrollo, nos encontramos ante el siguiente problema:

“Dado un fichero de conexiones almacenadas queremos calcular las conexiones contenidas en los diversas máquinas que se dan”

La primera pregunta que se nos plantea es: ¿Qué es una conexión contenida en otra?, estas conexiones se caracterizan por ser conexiones pero en las cuales el tiempo de inicio de la conexión que contiene a la otra es menor que el tiempo de inicio de esta segunda y el tiempo de fin de la conexión contenida es menor que el tiempo de fin que de la conexión que la contiene.

Para hacernos una idea gráfica mostraremos en la figura 1-1.3 la definición visual de conexión contenida:

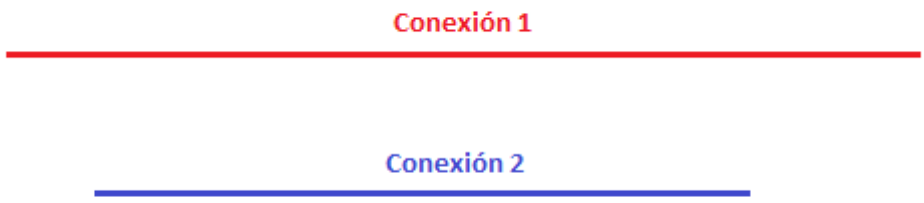

Figura 1-1.3: Definición de conexión contenida

Como podemos apreciar la conexión uno contiene a la conexión dos, porque tiene un tiempo de inicio anterior que la conexión dos y un tiempo de fin superior al de la conexión dos.

Anteriormente se ha indicado que las conexiones concurrentes son similares a las conexiones contenidas, esto es debido a que si relajamos la condición de tiempo de fin, es decir, la eliminamos, sobre las conexiones contenidas obtenemos la definición de conexión concurrente.

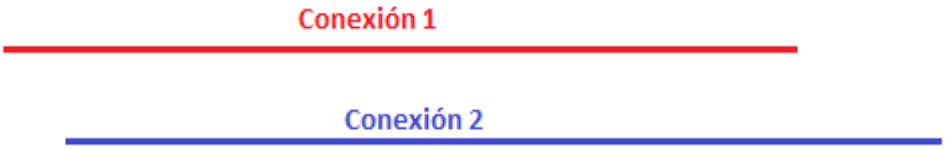

Figura 2-1.3 Definición de conexión concurrente

Como podemos ver la conexión uno sí que tiene un tiempo de inicio anterior al de la conexión dos, pero no tiene por qué tener un tiempo de fin superior al de la conexión dos, con esto se concluye fácilmente, para poder empezar el estudio que una vez se calculen las conexiones contenidas se pueden calcular las conexiones concurrentes

relajando la condición, y que además las conexiones contenidas están contenidas en el conjunto de las conexiones concurrentes.

1.4 Estructura del documento

Para abordar este problema, hemos decidido estructura este documento en varios apartados, los cuales nos permitirán comprender mejor los procesos de optimización que hemos seguido.

En el primer capítulo exponemos la introducción de nuestro problema, que consta de los siguientes apartadas:

Motivación: En este apartado explicaremos el porqué de nuestro problema, es decir, los motivos por los cuales es importantes el desarrollo del algoritmo que estamos haciendo.

Objetivo: En este apartado explicaremos los objetivos a los que se quieren llegar con este trabajo de fin de grado.

Contexto: En este apartado se explica el problema que se tiene y las definiciones que serán necesarias para poder comprender este trabajo.

Estructura del documento: En este apartado se muestra la estructura del documento

En el siguiente capítulo se tratará el coste teórico que tenemos sobre cada iteración, es decir, analizaremos de forma teórica la mejora que ha sufrido el algoritmo que estamos desarrollando, para ello lo hemos dividido en varias secciones:

Introducción: En esta sección se trata como vamos a desarrollar las siguientes secciones del capítulo.

Algoritmo de fuerza bruta: En esta sección trataremos el análisis teórico de nuestro primer algoritmo que será un algoritmo de fuerza bruta.

Algoritmo de fuerza bruta con eliminación de conexiones expiradas: En esta sección se tratará la primera iteración desarrollada de forma teórica de nuestro algoritmo, que es el desarrollo de un algoritmo por fuerza bruta.

Algoritmo de conexiones expiradas con eliminación de conexiones no validas: En este apartado se explicará el desarrollo teórico de la siguiente iteración que aplicamos a nuestro algoritmo, que es tras tener desarrollado el algoritmo por fuerza bruta eliminar las conexiones que no tienen sentido ver porque no tienen datos de interés.

Algoritmo de eliminación con ordenación de conexiones con respecto al tiempo: En este apartado se tratará de forma teórica la siguiente iteración del algoritmo que es ordenar las conexiones con respecto a su tiempo de fin, una vez hemos realizado la iteración.

Algoritmo final: En esta sección hacemos un desarrollo teórico de la última iteración dándonos el algoritmo final, en este caso lo que realizamos es que tras tener ordenadas las conexiones reducimos las conexiones que tenemos que analizar debido a cómo se generan las conexiones con esta ordenación.

Tras este capítulo teórico nos encontramos ante el capítulo 3, que trata el coste real de ejecución de nuestro algoritmo, que consta de las secciones siguientes:

Introducción: En esta sección introducimos los lenguajes que vamos a utilizar para desarrollar nuestro algoritmo de forma real.

Algoritmo de fuerza bruta: En esta sección observaremos si el algoritmo que denominamos como algoritmo de fuerza bruta tiene un coste real parecido al teórico calculado en el capítulo 2 en la sección de algoritmo de fuerza bruta.

Algoritmo de fuerza bruta con eliminación de conexiones expiradas: En este apartado trataremos la relación entre el coste teórico calculado en la sección 2 del capítulo 2 con el coste real de la implementación.

Algoritmo de conexiones expiradas con eliminación de conexiones no validas: En esta sección comprobaremos si el coste real está relacionado con el coste teórico calculado anteriormente.

Algoritmo de eliminación con ordenación de conexiones con respecto al tiempo: Tras los cálculos anteriores, procederemos en este apartado a ilustrar si nuestro coste teórico calculado es correspondiente con nuestro coste real.

Algoritmo final: Después de los apartados anteriores pasaremos a ver si nuestro algoritmo final tiene el mismo coste teórico estimado que el coste real.

Conclusiones: Por último en esta sección pasaremos a realizar una comparativa de todos los algoritmos utilizados con los otros algoritmos

Como último capítulo tenemos el capítulo que trata acerca de los estudios futuros así como las conclusiones, este capítulo consta de las secciones:

Conclusiones teóricas y prácticas: En este apartado se ilustrará en conjunto las conclusiones teóricas y prácticas alcanzadas en el proyecto.

Estudios de futuro y posibles mejoras: En esta sección se ilustrará de una forma teórica las mejoras que se podrían llevar acabo si se mejora el algoritmo de ordenación, que viene dato en la subsección "Cambio del algoritmo de ordenación", así como la adaptación del algoritmo en caso de querer utilizar multiproceso, que se explica una posible implementación en la subsección "Añadir multiproceso en el bucle del cálculo de las conexiones contenidas"

Este documento también consta de los siguientes anexos:

Anexo A: En este anexo se datan los campos que se utilizan en el desarrollo del programa, dándonos así una visión general de los datos que tendremos que analizar posteriormente.

Anexo B: En este anexo mostramos un ejemplo de como son los datos que nos da el snifer.

Anexo C: En este anexo mostramos un ejemplo de resultado, así como los campos que tienen los resultados.

2 Cálculo teórico del coste del algoritmo

2.1 Introducción

En esta sección nos preocuparemos de saber cómo abordar el problema por primera vez, así como de mostrar cómo es el fichero que recibimos del snifer.

El problema que se nos plantea es el análisis de conexiones contenidas en la red, entonces dado un fichero de nuestro snifer obtener las conexiones y decir cuales son contenidas. La restricción en la cual nos encontramos es realizar la ejecución en un tiempo inferior a un día, ya que se tiene que realizar un análisis diario de la topología de la red.

2.2 Algoritmo de fuerza bruta

Debido a que no se tienen datos del coste de cómputo de este problema primero nos decantamos por realizar un análisis por fuerza bruta, en que consiste esto: consiste en realizar el algoritmo más primitivo con el cuál se puede calcular las conexiones contenidas. Él porque realizamos esto es debido a que si el algoritmo por fuerza bruta tiene un coste muy bajo entonces directamente no hace falta realizar una implementación más compleja, ya que el mismo algoritmo nos serviría para conseguir nuestra meta. Para poder analizarlo más fácilmente mostraremos el pseudocódigo:

1 Leemos el fichero

2 Para cada conexión del fichero c1:

3 Para cada conexión del fichero c2:

4 Si la ip de c1 es igual a la ip de c2 y el puerto de c1 es igual al puerto de c2 entonces:

5 Si el tiempo de inicio de c1 es >= al de c2 y el tiempo de fin de c1 es >= que el de c2:

6 La conexión está contenida.

Como podemos ver la operación básica en este bucle es la línea 4, dentro de ella no se ejecuta ningún bucle ni ningún salto, en consecuencia dentro de la línea 4 inclusive esta misma tienen orden O(1).

Ahora bien tenemos N conexiones en el fichero, y dos bucles anidados que van de la conexión 0 a la N los dos, en consecuencia tenemos N^2 x O(1) lo que implica $O(N^2)$

Para ser más exactos tenemos una complejidad $3* N^2 + N$. Tras realizar las pruebas que se mostrarán en la siguiente sección se vio que se tenía que reducir la cota de este método para poder ser manejable.

En los siguientes pasos se verá cómo se reduce esta cota y los cambios que se siguen para llegar a ser un algoritmo más rápido. Para tener una idea aproximada la gráfica teórica es de la forma:

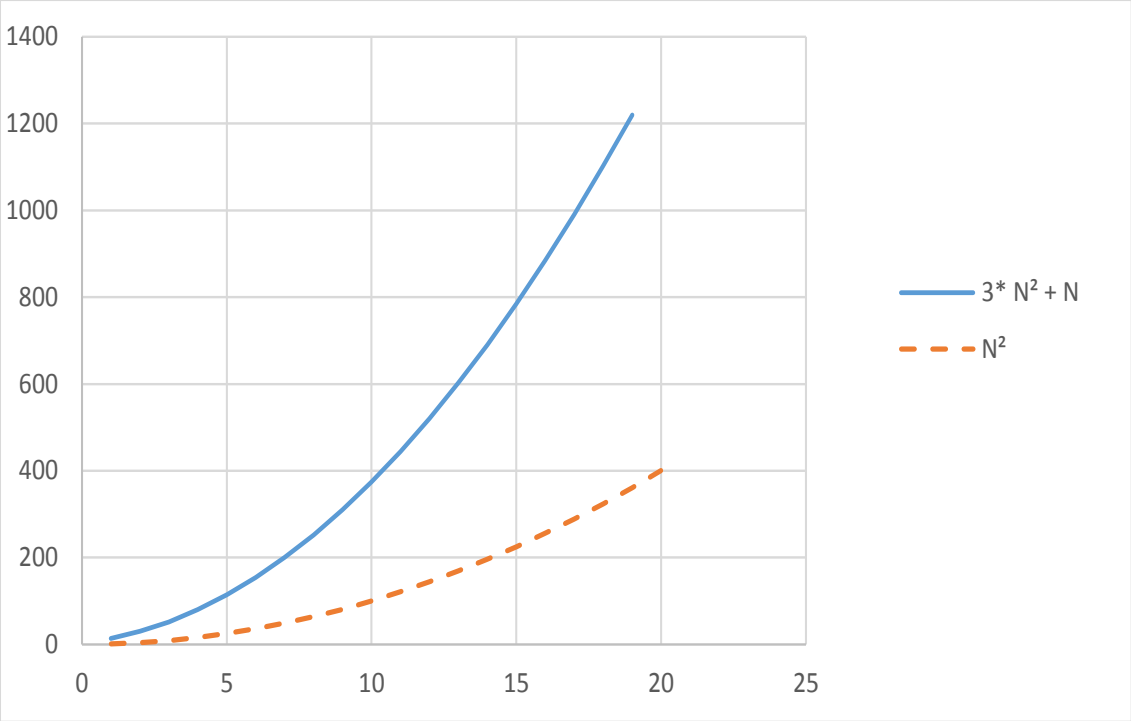

Figura 3-2.2: Estimación teórica del Algoritmo de fuerza bruta

2.3 Algoritmo de fuerza bruta con eliminación de conexiones expiradas

Al realizar el primer algoritmo se puede ver que no hace falta analizar todas las conexiones ya que según expira el tiempo de esta conexión ya no hace falta analizarla, debido a que ya no puede estar contenida dentro de las siguientes. Recordando el concepto, se da que una conexión está contenida en otra si el tiempo de inicio de una es

mayor que el tiempo de inicio de la otra y si el tiempo de fin de la contenedora es mayor que el de la contenida. Por lo tanto lo que hacemos en nuestro algoritmo nuevo lo que hacemos es marcar que conexión es la primera que no ha expirado, es decir, que el tiempo de fin de la conexión que estamos analizando es menor que el tiempo de inicio de la que queremos ver si está contenida. Por lo tanto realizamos lo siguiente:

1 Leemos el fichero

2 Conexión expirada = 0

3 Para cada conexión del fichero c1 del fichero:

3 Desde c2=conexión expirada hasta la última conexión:

4 Si la ip de c1 es igual a la ip de c2 y el puerto de c1 es igual al puerto de c2 entonces:

5 Si el tiempo de inicio de c1 es >= al de c2 y el tiempo de fin de c1 es >= que el de c2:

6 La conexión está contenida.

7 Desde conexión expirada hasta conexión c2 (i):

8 Si conexiones borradas tiempo de inicio < que el tiempo de fin de la conexión i:

9 conexión expirada +=1

Como podemos observar parece que la iteración no ha reducido, de hecho podría llegarse a pensar que ha aumentado, pero en realidad estamos en el mismo orden y además estamos reduciendo el número de conexiones que analizamos en este segundo bucle, ya que ya no tenemos N^2 si no que eliminamos según pasa el tiempo en el fichero. Después de este cambio se puede ver que el número de iteraciones del segundo bucle pasa a ser menor que en el algoritmo anterior. Esto implica que estamos reduciendo el factor de escala que teníamos anteriormente, es decir, que de un número de operaciones igual a $3* N^2 + N$ pasa a ser un valor menor que 3.

Como podemos ver el número de conexiones que tenemos que analizar con respecto al anterior es menor, y basándonos en la media de conexiones descartadas bajo un experimento práctico, en el cuál ejecutamos varios ficheros con el mismo número de

conexiones. Tenemos que en media el número de conexiones que han expirado entre el total de conexiones vemos que obtenemos un factor de 0.4998921, lo que hace que nuestro segundo bucle se ejecute la mitad de veces.

Por lo tanto tenemos que de un número de iteraciones $3* N^2 + N$ pasamos a tener $3* N*(0.4998921*N) + N= 1.499676* N^2 + 2N$, el 2N anterior es debido a las iteraciones que tiene que realizar las conexiones para borrarlas más la lectura del fichero. Con esto podemos observar que de forma teórica hemos reducido el número de iteraciones a más de la mitad. Dado que ya hemos reducido este factor de escala pasaremos a llamar a nuestro algoritmo de fuerza bruta con eliminación de conexiones expiradas algoritmo de conexiones expiradas.

Para tener una idea aproximada de cómo ha cambiado nuestra función tendremos que observar la figura 4-2.3:

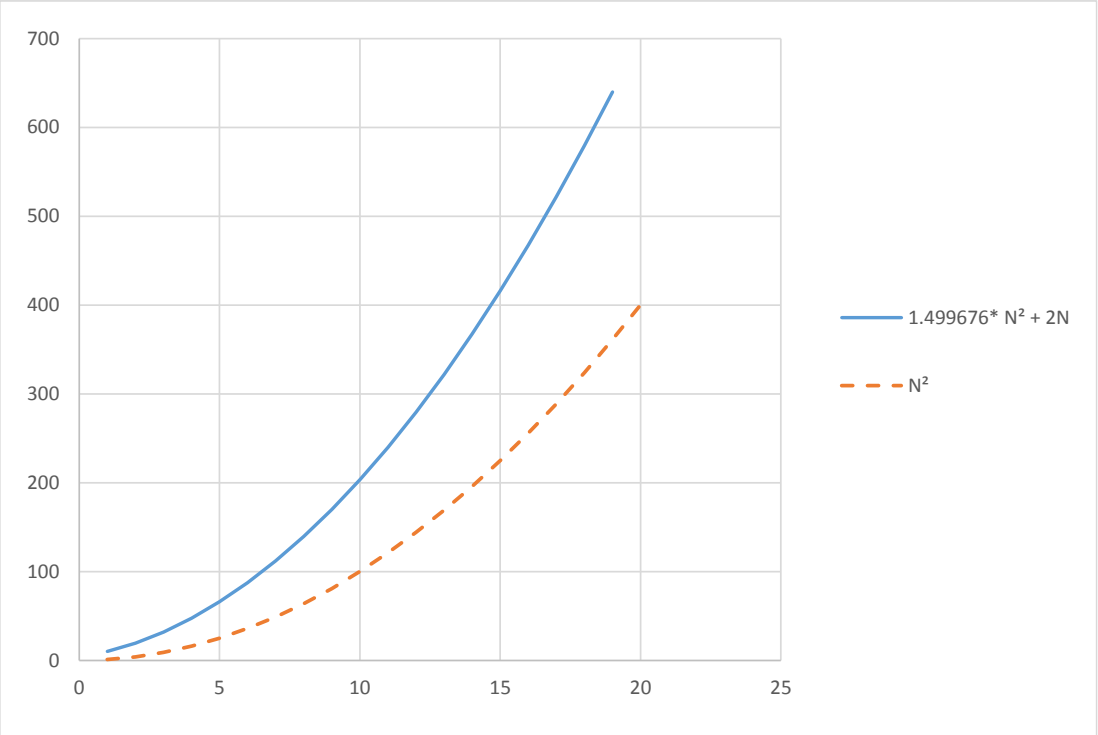

Figura 4-2.3: Tiempo teórico del algoritmo de fuerza bruta con eliminación de conexiones

2.4 Algoritmo de conexiones expiradas con eliminación de conexiones no validas

Tras analizar un poco nuestro fichero y debido a que el flujo en internet es generalmente constante nos damos cuenta que cuando conectamos el snifer ya se encuentran conexiones establecidas, en consecuencia nos encontramos con conexiones que tan siquiera son válidas, ya que han empezado antes de nuestra captura, así como hay conexiones cuando paramos el snifer aún prosiguen con su ejecución. Por lo tanto esas conexiones como las detectamos pues son aquellas que tienen los campos del 17 al 23 iguales a 0. Estos campos son:

Número	Nombre	Descripción
17	numberPacketSrcToDst	Número de paquetes entre cliente y servidor
18	numberPacketDstToSrc	Número de paquetes de servidor a cliente
19	numberSYNSrcToDst	Número de Syn's de cliente a servidor
20	numberSYNDstToSrc	Número de Syn's de servidor a cliente
21	numberFINSrcToDst	Número de FIN's de cliente a servidor
22	numberFINDstToSrc	Número de FIN's de servidor a cliente

Tabla 1: Campos utilizados para eliminar las conexiones no validas

Si algunos de estos campos es 0 entonces tenemos el caso en que esa conexión ha sido producida erróneamente, que sean mayores que ceros y distintos no implica que sean erróneas, esto es debido a que se pueden haber perdido paquetes y estos haber sido replicados por este motivo, no podemos descartar las conexiones con números de Syncs y Fins distintos. Pasemos a escribir y analizar nuestro pseudocódigo:

1 Para cada línea del fichero

2 si el campo 17 o el 18 o el 19 o el 20 o el 21 o el 22 == 0 entonces

3 descartar conexión

4 Conexión expirada = 0

5 Para cada conexión del fichero c1 del fichero:

6 Desde c2=conexión expirada hasta la última conexión:

7 Si la ip de c1 es igual a la ip de c2 y el puerto de c1 es igual al puerto de c2 entonces:

8 Si el tiempo de inicio de c1 es >= al de c2 y el tiempo de fin de c1 es >= que el de c2:

9 La conexión está contenida.

10 Desde conexión expirada hasta conexión c2 (i):

11 Si conexiones borradas tiempo de inicio < que el tiempo de fin de la conexión i:

12 conexión expirada +=1

Una vez descartamos estas conexiones nos encontramos con una reducción del número de conexiones a analizar sobre nuestro fichero, aproximadamente se elimina un 33% de las conexiones que tenemos que analizar, por lo tanto el orden del algoritmo nos cambia de la siguiente forma:

Nuestro valor anterior era: $1.499676* N^2 + 2N$, entonces leemos el mismo número de conexiones que el anterior N pero tenemos que el bucle interno de orden N^2 ahora se ejecuta en vez de N^2 veces se ejecuta $(0.66*N)^2$ por lo tanto tenemos:

$1.499676* (0.66*N)^2 + 2N= 2N+0.653259* N^2$ lo que hace una reducción de la cota en un 129.56%, pero falta sumarle el aumento del número de instrucciones que ejecutamos, en vez de N en la lectura es 2N por lo tanto tenemos un total de 3*N del orden lineal, dando un resultado total de $3N+0.653259* N^2$ que es una mejora con respecto al tiempo anterior.

Prosigamos observando como nuestra curva va haciéndose más pequeña con respecto a la anterior:

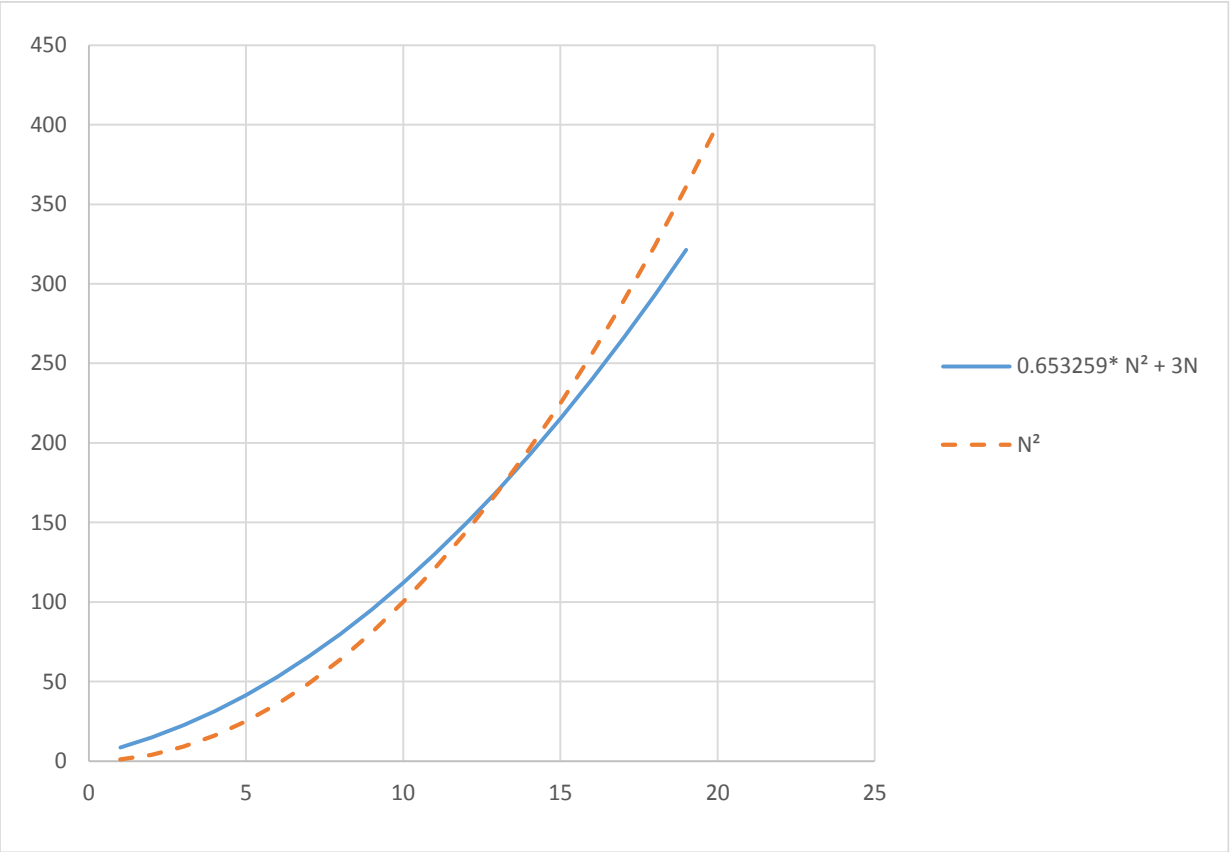

Figura 5-2.4: Estimación teórica del Algoritmo de conexiones expiradas con eliminación de conexiones no validas

Después de realizar esta reducción en el número de líneas a analizar llamaremos a nuestro algoritmo nuevo Algoritmo de eliminación.

2.5 Algoritmo de eliminación con ordenación de conexiones con respecto al tiempo

Una vez tenemos nuestro algoritmo de eliminación nos planteamos como podríamos mejorar la velocidad del mismo, y cómo podemos ver tenemos una variable que borra las conexiones que ya no son necesarias en nuestro análisis, pero claramente y debido a que podemos tener una conexión muy larga y esta no es descartada hasta pasado un gran lapso de tiempo, podemos solucionar este problema ordenando las conexiones por tiempo de inicio, y desempatando en tiempo de fin.

La pregunta que nos podemos plantear es ¿Por qué mejoraría esto la velocidad?, y la respuesta a esta pregunta es, que debido a como están distribuidas en el fichero es

posible que una conexión con tiempo de inicio y tiempo de fin muy separados, es decir una conexión que ha estado mucho tiempo se mantendría durante mucho tiempo como primera conexión a analizar con las siguiente, porque su tiempo de fin hace que esta se borre mucho más tarde que una mucho más corta, entonces lo que nos interesa que sea de las primeras en ser analizadas para poder borrarla cuanto antes mejor, ya que al ser tan duradera a lo largo del tiempo va a tener muchas conexiones contenidas. Para ello utilizaremos el algoritmo de ordenación Quicksort, sobre las conexiones que son válidas para nuestro algoritmo posterior de cálculo de conexiones contenidas. Quedando el siguiente algoritmo:

1 Para cada línea del fichero

2 si el campo 17 o el 18 o el 19 o el 20 o el 21 o el 22 == 0 entonces

3 descartar conexión

4 quicksort(conexiones, función de ordenación)

5 Conexión expirada = 0

6 Para cada conexión del fichero c1 del fichero:

7 Desde c2=conexión expirada hasta la última conexión:

8 Si la ip de c1 es igual a la ip de c2 y el puerto de c1 es igual al puerto de c2 entonces:

9 Si el tiempo de inicio de c1 es >= al de c2 y el tiempo de fin de c1 es >= que el de c2:

10 La conexión está contenida.

11 Desde conexión expirada hasta conexión c2 (i):

12 Si conexiones borradas tiempo de inicio < que el tiempo de fin de la conexión i:

13 conexión expirada +=1

Como podemos ver a priori nuestro tiempo de ejecución puede parecer que ha aumentado ya que ahora tenemos un factor O(N*log(N)) que antes no aparecía pero en

realidad nos ha facilitado el cálculo de las conexiones contenidas, ya que de un esquema como el que aparece en la figura 6-2.5:

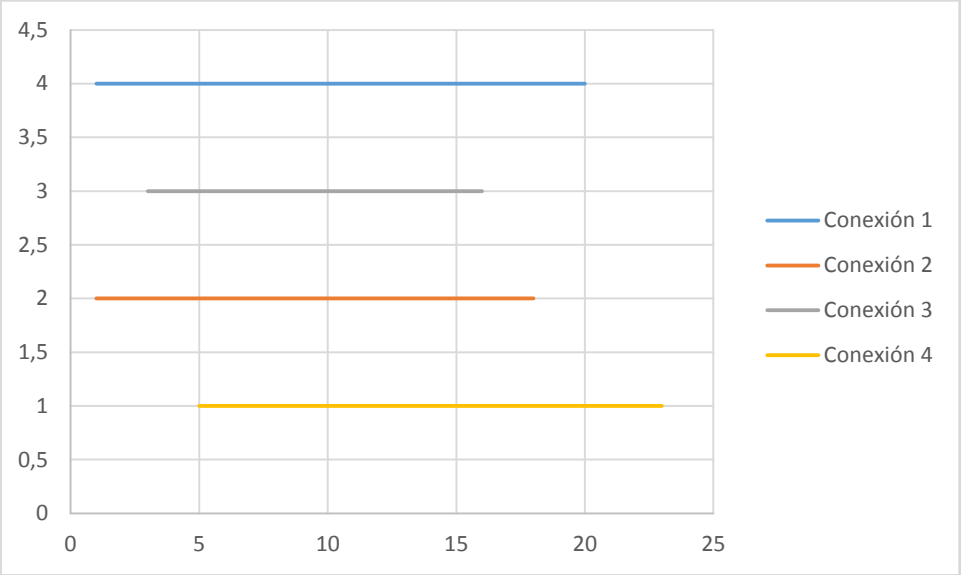

Figura 6-2.5: Estado anterior a Quicksort de las conexiones

Pasamos a este tipo de esquema que se encuentra en la figura 7-2.5:

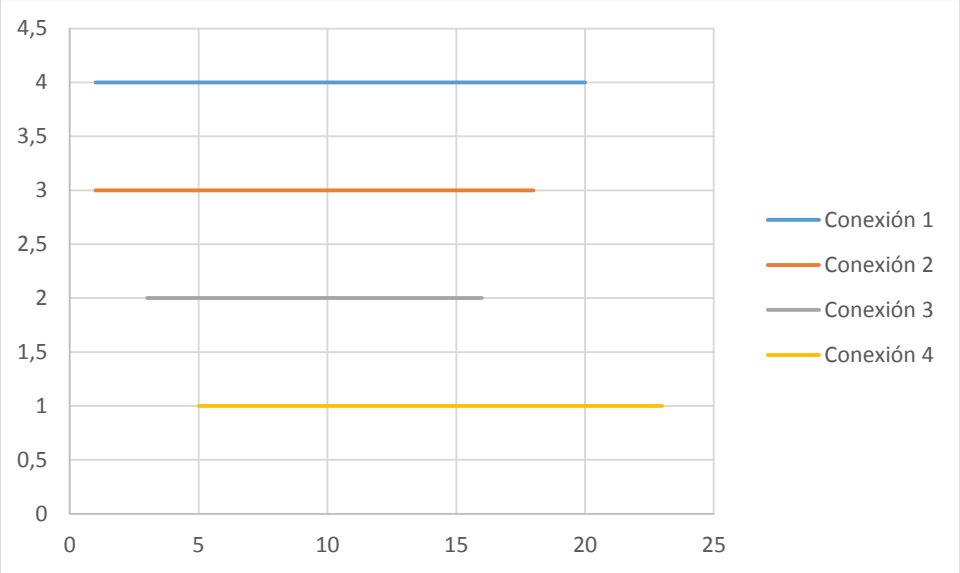

Figura 7-2.5: Estado posterior a Quicksort de las conexiones

Entonces a la hora del segundo bucle de borrado de conexiones en seguida las conexiones un poco más grandes van a ser borradas mucho más rápido, lo cual hace una mejora no muy elevada debido al coste de quicksort pero si una mejora en cuanto al

rendimiento del algoritmo ya que las conexiones contenidas con tiempos muy largos son analizadas antes y enseguida se eliminan. Aun así de forma teórica tendríamos un orden de $3N+0.653259* N^2 + 0.653259*N*\log(0.6553259*N)$ lo que hace que aun así nuestro algoritmo sea más rápido que N^2 como podemos ver en la figura 8-2.5

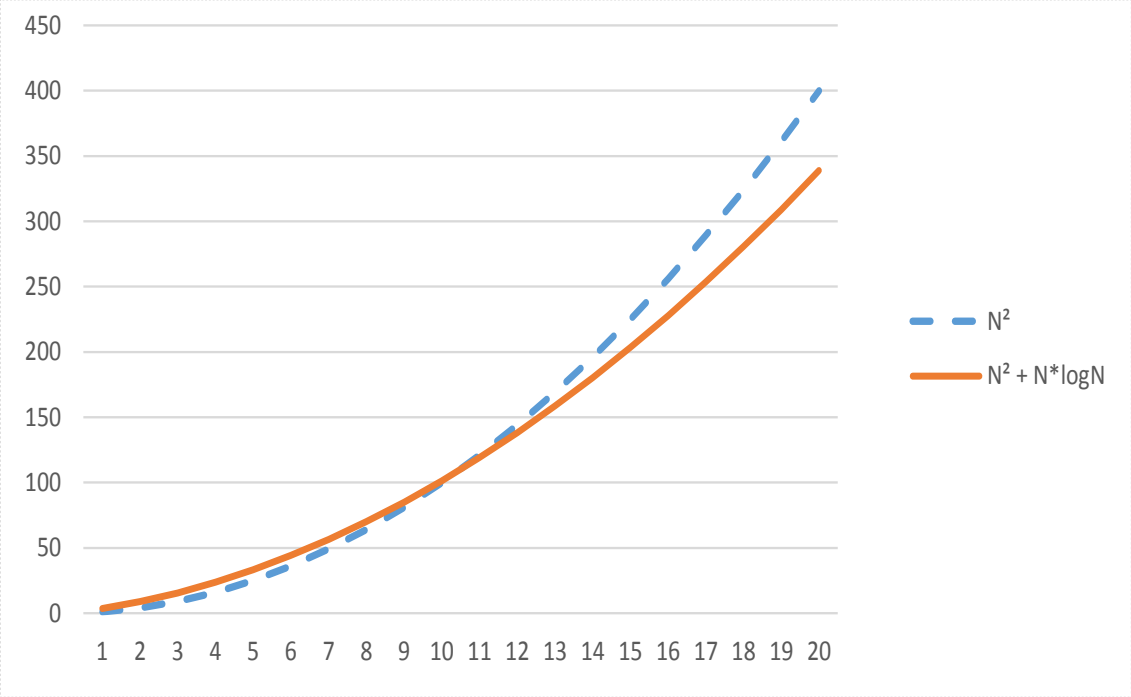

Figura 8-2.5: Tiempo teórico utilizado QuickSort

Como podemos ver en un principio no parece una mejora con respecto al anterior, pero más adelante con la última iteración veremos que si puede mejorar dando este paso intermedio. Se ha decidido utilizar Quicksort dado que en temas referentes al acceso de ficheros Quicksort funciona mejor que Mergesort, aunque tienen el mismo orden.

Por último pasaremos a llamar a nuestro algoritmo de eliminación con ordenación a algoritmo de eliminación ordenada.

2.6 Algoritmo final

Una vez tenemos nuestro algoritmo de eliminación ordenada, ya podemos pasar a la última mejora del algoritmo que vamos a analizar, aunque este mismo se puede mejorar todavía más, se datará más adelante la forma de mejorarlo.

Ahora bien la forma de mejorar esta última iteración es darse cuenta que en realidad no es necesario empezar en la conexión borrada si no en la conexión i-esima en

la que nos encontramos, ya que una vez están ordenadas las conexiones contenidas se sitúan en cascada, por lo tanto las contenidas de la i-esima estarán entre la i-esima y hasta que se acabe la conexión.

En consecuencia si además añadimos a su vez que el bucle en vez de parar en la última conexión pare en la conexión en la cual el tiempo de fin de la i-esima sea menor que el de esta conexión.

1 Para cada línea del fichero

2 si el campo 17 o el 18 o el 19 o el 20 o el 21 o el 22 == 0 entonces

3 descartar conexión

4 quicksort (conexiones, función de ordenación)

5 Para cada conexión del fichero c1 del fichero:

6 Desde c1 hasta la última conexión o hasta que el tiempo de inicio de c2 > tiempo de inicio de c1:

7 Si la ip de c1 es igual a la ip de c2 y el puerto de c1 es igual al puerto de c2 entonces:

8 Si el tiempo de inicio de c1 es >= al de c2 y el tiempo de fin de c1 es >= que el de c2:

9 La conexión está contenida.

Puede parecer que no se ha mejorado mucho, pero en realidad ha habido un cambio de orden en la velocidad del algoritmo, esto es debido a que un servidor solo puede tener M conexiones contenidas a la vez, el número de veces que se va a iterar el segundo bucle es máx {M,N-i} porque una vez se alcance M ya no habrá más conexiones contenidas posibles y además como máximo se ejecutará M * N veces el bucle quedando por tanto en vez de un factor de $O(N^2)$ un factor de O(N*log(N)) lo que hace que el algoritmo para un número de conexiones muy elevado sea más rápido.

Puede parecer como que la cota se ha elevado entonces sea más lento pero en caso de haber muchas conexiones, es decir, analicemos una red con más de 10 millones de

conexiones, que en general, en un día todas las redes suelen tener más de ese número de conexiones el algoritmo ejecutará más rápido.

Para ser más claro supongamos que tenemos 4 conexiones ya ordenadas, como en la figura 7-2.5, supongamos también que un servidor solo puede tener como máximo 3 conexiones contenidas. También supongamos que no se realizan conexiones con cualquier otro ordenador para no dificultar nuestra compresión.

Entonces como solo puede tener 3 como máximo lo que tendremos que iterar como máximo 3 veces por cada conexión porque no puede tener más a la vez luego, o tiene 3 o tiene menos, luego para cada conexión tendremos que mirar a lo largo de su tiempo de vida 3 conexiones siguiente. En consiguiente nuestros dos bucles se ejecutaran 3*N veces. Así pues el orden final pasará a ser N*log(N) ya que es la mayor iteración que debemos hacer. Como estimación teórica tenemos la siguiente figura:

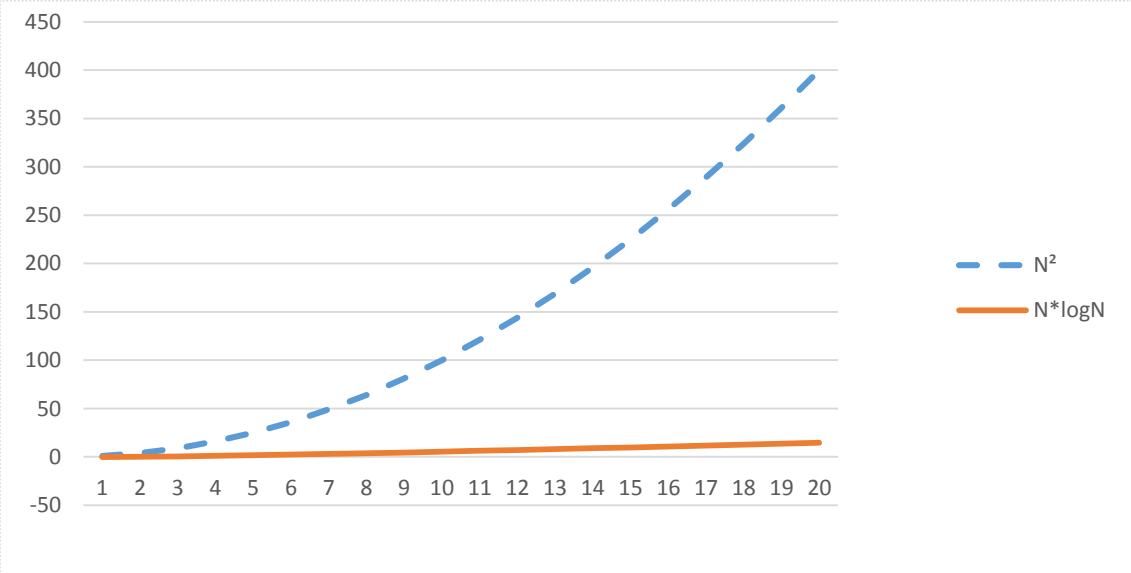

Figura 9-2.6: Tiempo teórico del algoritmo final

Por último y cómo podemos ver en la figura 9-2.6 la mejora en el tiempo de ejecución es más que sustancial.

2.7 Conclusión

Como conclusión obtenemos que, de forma teórica, nuestro algoritmo tendrá un coste de orden O(N*log(N)) que como ya sabemos con anterioridad por la figura 1-2.6 es bastante mejor que el algoritmo de fuerza bruta de orden $O(N^2)$. Para ver que esto tiene sentido pasaremos a un estudio práctico que aparece en el siguiente capítulo.

3 Coste real del algoritmo

3.1 Introducción

En este capítulo vamos a ilustrar con los diversos programas si los costes teóricos de este algoritmo desarrollado son los reales, para ello realizaremos una implementación del algoritmo teórico y veremos si las gráficas son correspondiente con el tiempo teórico, o tienen una forma similar.

Para ello se han utilizado para las primeras iteraciones AWK y Shell para la realización de las pruebas, y posteriormente se pasó a un lenguaje compilado que es C. Este cambio se debe a que los lenguajes interpretados son más lentos, y las pruebas se demoraban demasiado y se decidió cambiar debido a que los lenguajes compilados ejecutan más rápido que los interpretados.

Puede pensarse que esto no es del todo cierto porque AWK tiene una gran parte implementada en C, pero aun así las llamadas en Shell son mucho más lentas que un lenguaje compilado como C.

Por otro lado se ha de saber que ha excepción de las pruebas realizadas con el fichero de 3GB que se han realizado sobre otra máquina la cual desconozco las especificaciones. Las pruebas de rendimiento sobre el resto de ficheros incluyendo la del fichero de 80MB se han realizado sobre un ordenador con procesador i3 de primera generación a 2.7GHz y 4GB de RAM

Los campos del fichero que vamos a utilizar vienen datados en el Anexo A, por si se quiere tener un ejemplo de cómo son las líneas del fichero sobre el que se ejecutan estas pruebas se tiene el anexo B. Por último si se quiere ver el formato de salida que se utiliza en el programa se tiene el anexo C.

3.2 Algoritmo de fuerza bruta

En este apartado se ha realizado una implementación del algoritmo de fuerza bruta que aparece en el capítulo 2, cuyo pseudocódigo es:

1 Leemos el fichero

2 Conexión expirada = 0

3 Para cada conexión del fichero c1 del fichero:

3 Desde c2=conexión expirada hasta la última conexión:

4 Si la ip de c1 es igual a la ip de c2 y el puerto de c1 es igual al puerto de c2 entonces:

5 Si el tiempo de inicio de c1 es >= al de c2 y el tiempo de fin de c1 es >= que el de c2:

6 La conexión está contenida.

7 Desde conexión expirada hasta conexión c2 (i):

8 Si conexiones borradas tiempo de inicio < que el tiempo de fin de la conexión i:

9 conexión expirada +=1

Como estamos utilizando AWK la lectura del fichero va implícita en la ejecución del programa ya que AWK tiene tres fases de ejecución BEGIN que es antes que se empiece a leer el fichero, luego tenemos el cuerpo del programa mientras se va leyendo el fichero, y por último tenemos el apartado de END, el cual se ejecuta después de la lectura del fichero quedando el siguiente coste:

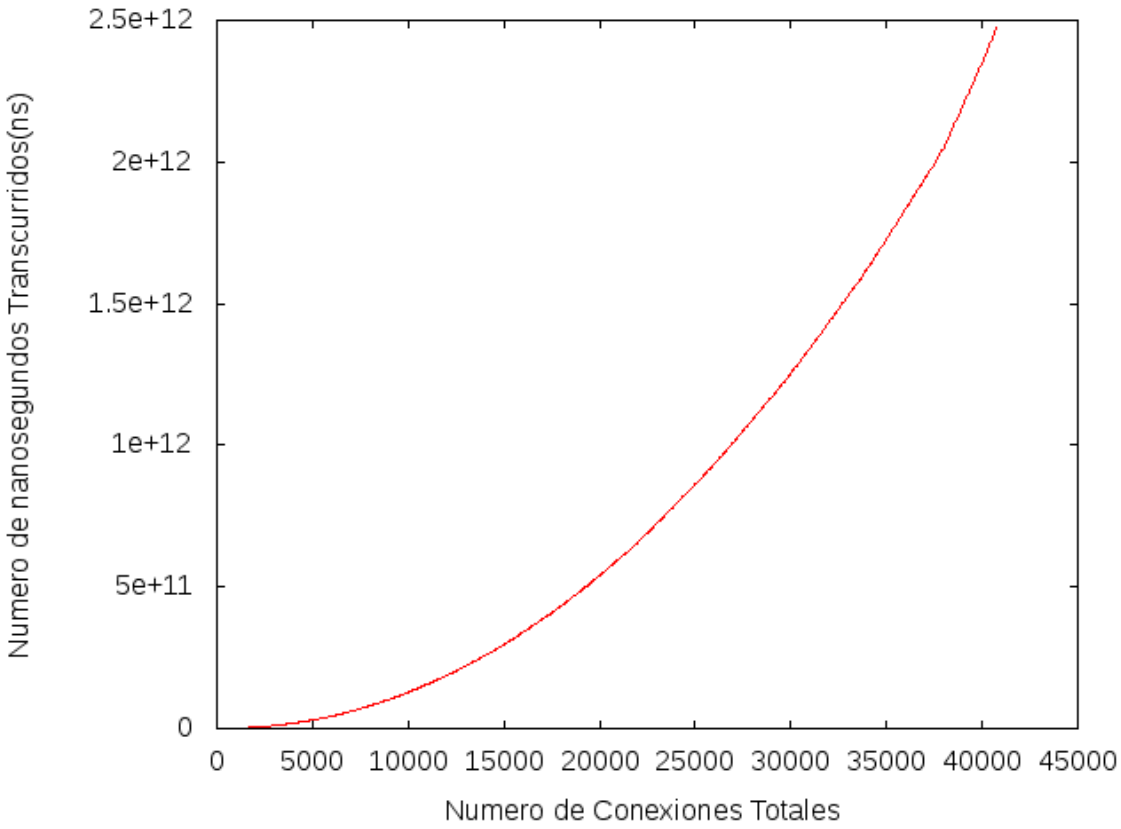

Figura 10-3.2: Coste Real del algoritmo de fuerza bruta

Como podemos observar, el tiempo teóricos estimados y el tiempo de ejecución que se obtiene son muy parecidos, pero claramente es insuficiente para utilizarlo como un programa favorable, ya que en cuanto pasamos a un fichero real de 3GB que son aproximadamente 5.132.673 conexiones este tiempo pasa a ser superior a un día y para realizar estudios diarios no es un tiempo razonable, porque tendremos encolados muchas de estas ejecuciones. De hecho para hacernos una idea esta gráfica no tiene todas las conexiones del fichero de prueba porque el tiempo de ejecución era superior a 40 minutos con 80.8 MB

3.3 Algoritmo de fuerza bruta con eliminación de conexiones expiradas

Ahora vamos a comparar nuestro algoritmo de fuerza bruta con eliminación de conexiones expiradas implementado con AWK con respecto a su tiempo teórico.

Recordemos que el pseudocódigo de este programa que tenemos que realizar es:

1 Leemos el fichero

2 Conexión expirada = 0

3 Para cada conexión del fichero c1 del fichero:

3 Desde c2=conexión expirada hasta la última conexión:

4 Si la ip de c1 es igual a la ip de c2 y el puerto de c1 es igual al puerto de c2 entonces:

5 Si el tiempo de inicio de c1 es >= al de c2 y el tiempo de fin de c1 es >= que el de c2:

6 La conexión está contenida.

7 Desde conexión expirada hasta conexión c2 (i):

8 Si conexiones borradas tiempo de inicio < que el tiempo de fin de la conexión i:

9 conexión expirada +=1

Como ya sabemos el coste teórico de este algoritmo es: $1.499676 * N^2 + 2N$, pasemos entonces a analizar nuestro coste teórico con respecto a nuestro coste de la implementación. Tras realizar la implementación nos encontramos ante esta gráfica:

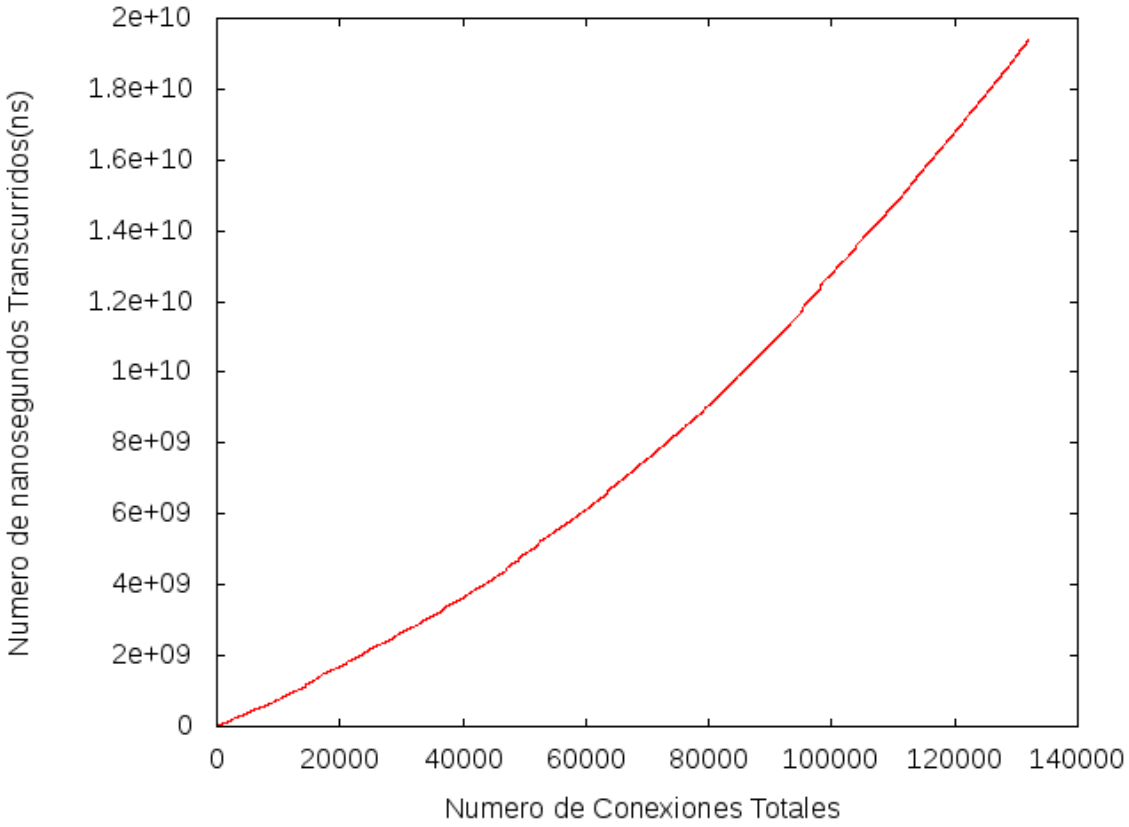

Figura 11-3.3: Resultado práctico del algoritmo de fuerza bruta con eliminación de conexiones expiradas

Como podemos ver nuestro coste de ejecución es muy parecido al coste teórico aunque con variaciones leves debido a la ejecución del ordenador. En este momento seguimos por encima de las 24 horas de ejecución por lo tanto sigue sin ser suficiente la reducción que se le ha aplicado al algoritmo, pero como ya hemos visto anteriormente en el capítulo 2 se puede reducir bastante más el tiempo de ejecución teórico y por consiguiente el práctico.

Como podemos observar en nuestra figura anterior con respecto a la figura 10-3.2 es que esta tarda en ejecutar alrededor de 20 segundos el fichero de 80.8 MB y en cambio el otro algoritmo, (el algoritmo de fuerza bruta), está tardando más de 41 minutos en calcular solo las 40000 primeras conexiones

.

3.4 Algoritmo de conexiones expiradas con eliminación de conexiones no validas

Ahora vamos a analizar la siguiente iteración que se ha realizado en nuestro programa. Este programa está escrito en AWK, en este punto es cuando ya vemos que tendremos que cambiar de lenguaje, aunque la confirmación viene tras la gráfica de tiempos siguiente. Procedamos a recordar el pseudocódigo descrito en la sección anterior:

1 Para cada línea del fichero

2 si el campo 17 o el 18 o el 19 o el 20 o el 21 o el 22 == 0 entonces

3 descartar conexión

4 Conexión expirada = 0

5 Para cada conexión del fichero c1 del fichero:

6 Desde c2=conexión expirada hasta la última conexión:

7 Si la ip de c1 es igual a la ip de c2 y el puerto de c1 es igual al puerto de c2 entonces:

8 Si el tiempo de inicio de c1 es >= al de c2 y el tiempo de fin de c1 es >= que el de c2:

9 La conexión está contenida.

10 Desde conexión expirada hasta conexión c2 (i):

11 Si conexiones borradas tiempo de inicio < que el tiempo de fin de la conexión i:

12 conexión expirada +=1

Pasemos entonces al estudio de la velocidad del algoritmo con respecto a su tiempo teórico.

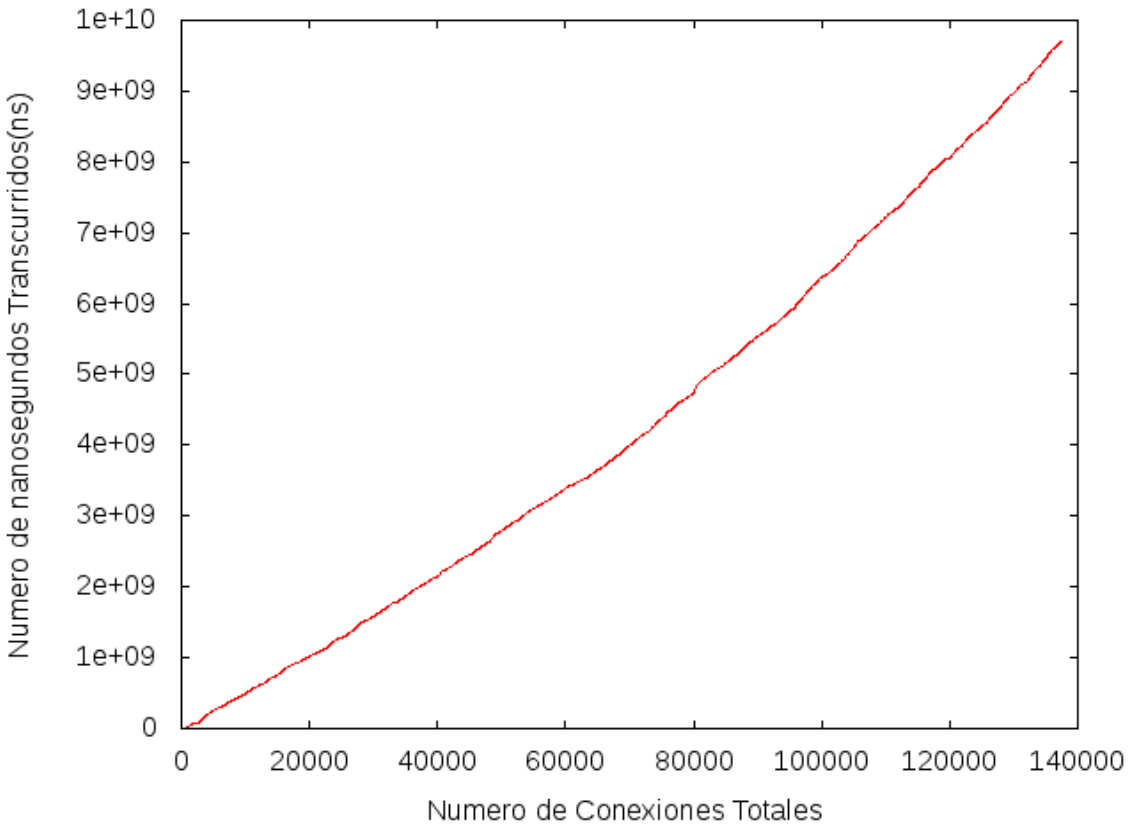

Figura 12-3.4: Resultado práctico del algoritmo de eliminación de conexiones con eliminación de no validas

Como podemos observar, el tiempo teórico sigue siendo correspondiente con respecto al tiempo práctico, como era de esperar. Ahora bien tras esta iteración sí se consiguió que el fichero de 3GB fuese ejecutado en menos de un día, lo que ocurre es que aún es mejorable, ya que la idea del uso de Quicksort apareció mientras se implementaba este apartado.

Tampoco es que se tenga una mejora sustancial con respecto a la figura 11-3.3, ya que como predijimos anteriormente la reducción sería en aproximadamente un 50%, y cómo podemos ver el tiempo final se ha reducido en aproximadamente eso.

3.5 Algoritmo de eliminación con ordenación

Una vez realizado el proceso anterior pasamos a utilizar Quicksort, y también cambiamos de lenguaje, ya que el Quicksort de C es muy rápido. Ahora bien en un principio esta iteración se sabía que iba a ir un poco más rápida por el cambio de lenguaje interpretado a lenguaje compilado, pero no lo suficientemente rápida como para considerarse una mejora sustancial en lo utilizado. Por otro lado mejorará un poco debido a que las conexiones expiradas se actualizan un poco más rápido pero no es una mejora en rendimiento muy elevada.

Para ser consistente con los demás apartados refresquemos el pseudocódigo que estamos analizando.

1 Para cada línea del fichero

2 si el campo 17 o el 18 o el 19 o el 20 o el 21 o el 22 == 0 entonces

3 descartar conexión

4 quicksort(conexiones, función de ordenación)

5 Conexión expirada = 0

6 Para cada conexión del fichero c1 del fichero:

7 Desde c2=conexión expirada hasta la última conexión:

8 Si la ip de c1 es igual a la ip de c2 y el puerto de c1 es igual al puerto de c2 entonces:

9 Si el tiempo de inicio de c1 es >= al de c2 y el tiempo de fin de c1 es >= que el de c2:

10 La conexión está contenida.

11 Desde conexión expirada hasta conexión c2 (i):

12 Si conexiones borradas tiempo de inicio < que el tiempo de fin de la conexión i:

13 conexión expirada +=1

Ahora bien tras la implementación de este pseudocódigo obtenemos esta estimación con respecto al tiempo teórico

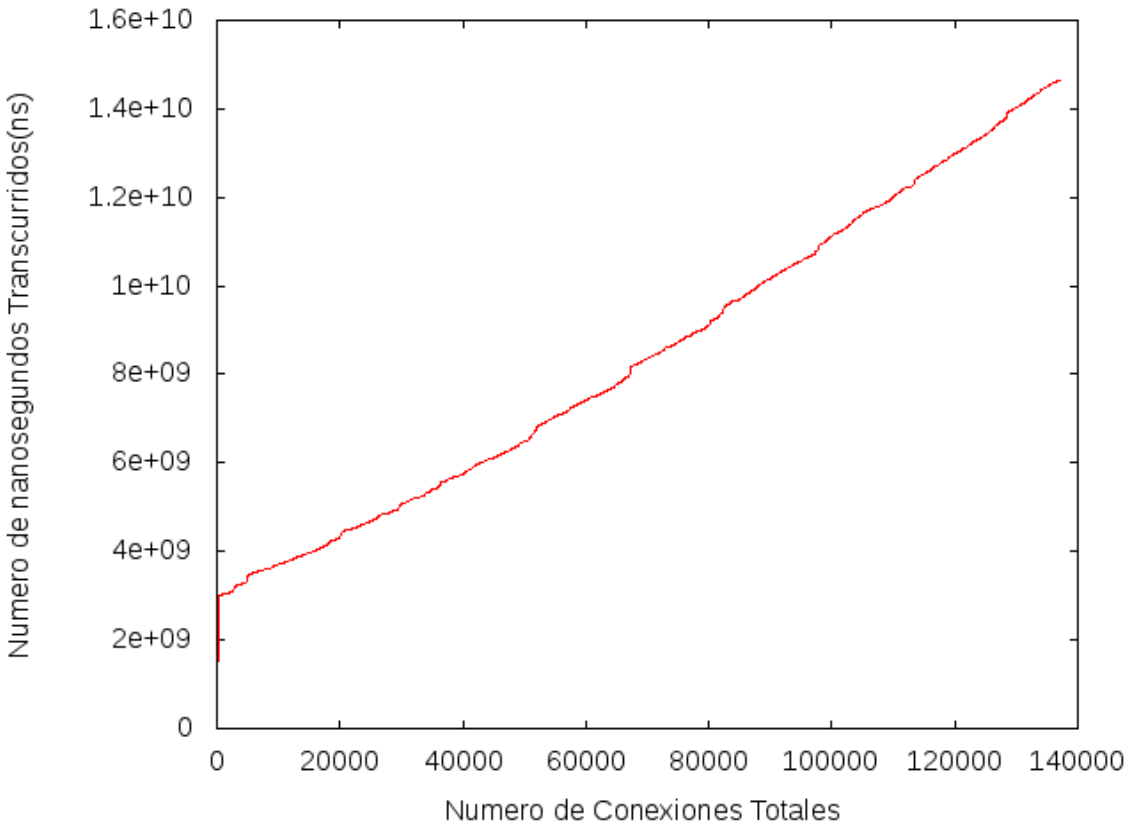

Figura 13-3.5: Tiempo práctico del algoritmo de eliminación con ordenación

Como podemos ver, en esta gráfica se puede apreciar el coste de quicksort, esto es debido a que el orden de Quicksort es cercano al coste de un doble bucle (N^2), pero aun así se sigue viendo que la función es cuadrática.

La ejecución con un fichero de 3GB no fue altamente satisfactoria ya que era algo superior a las 8 horas lo que implicaba que no fue una buena mejora para haber sido un cambio tan drástico como es el paso de interpretado a compilado.

Con respecto a la mejora realizada, como podemos ver en la figura anterior, y como se estimó no hay una mejora de tiempos pero si se puede ver cómo va apareciendo algo que podría ser cercano a N*log(N).

Como se puede apreciar anteriormente con respecto a la figura 13-3.5 y a la figura 12-3.4, no ha habido mejora de tiempos, pero bien es cierto que con este cambio se ve que se puede llegar a realizar el cambio de orden, escogiendo bien la condición del bucle.

3.6 Algoritmo final

Por último pasamos a la iteración final, esta iteración está realizada en C, en este programa se ha realizado una implementación del algoritmo final descrito en el capítulo 2. A continuación mostramos el pseudocódigo que se utilizó en el programa para realizar este programa:

1 Para cada línea del fichero

2 si el campo 17 o el 18 o el 19 o el 20 o el 21 o el 22 == 0 entonces

3 descartar conexión

4 quicksort (conexiones, función de ordenación)

5 Para cada conexión del fichero c1 del fichero:

6 Desde c1 hasta la última conexión o hasta que el tiempo de inicio de c2 > tiempo de inicio de c1:

7 Si la ip de c1 es igual a la ip de c2 y el puerto de c1 es igual al puerto de c2 entonces:

8 Si el tiempo de inicio de c1 es >= al de c2 y el tiempo de fin de c1 es >= que el de c2:

9 La conexión está contenida.

Una vez tenemos esto, y hemos comprendido el desarrollo teórico citado en el capítulo 2 podemos pasar a la comprobación del enunciado de manera práctico.

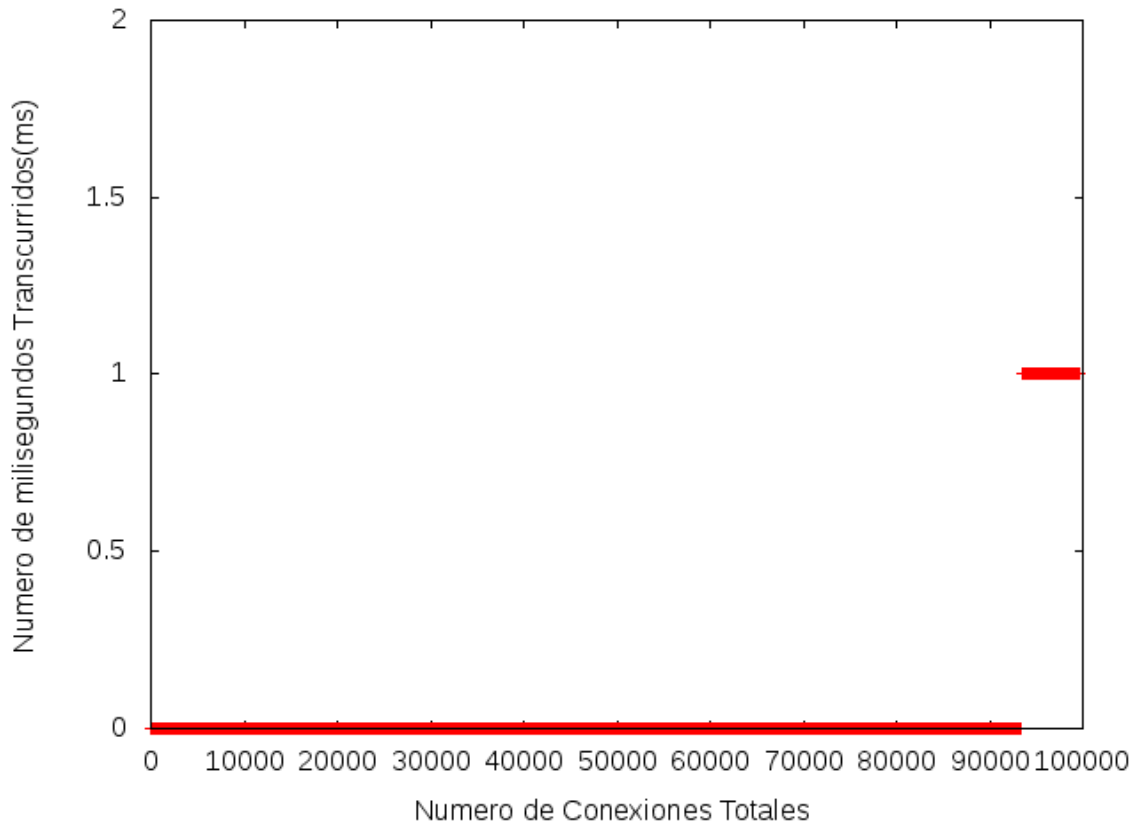

Figura 14-3.5: Tiempo Práctico del algoritmo final

Como podemos observar en esta gráfica el tiempo se ha reducido tanto que es prácticamente imperceptible el cambio entre el número de conexiones, es por esto que si nos damos cuenta estamos hablando del orden de segundos con los algoritmos de 80 MB y aquí en cambio estamos hablando de menos de milisegundos, con esto es claro que el orden entre los algoritmos anteriores y este no es el mismo.

Dado que no es muy apreciable en este fichero se realizó la prueba con otro fichero y poniéndole pausas para así poder ver mejor el orden

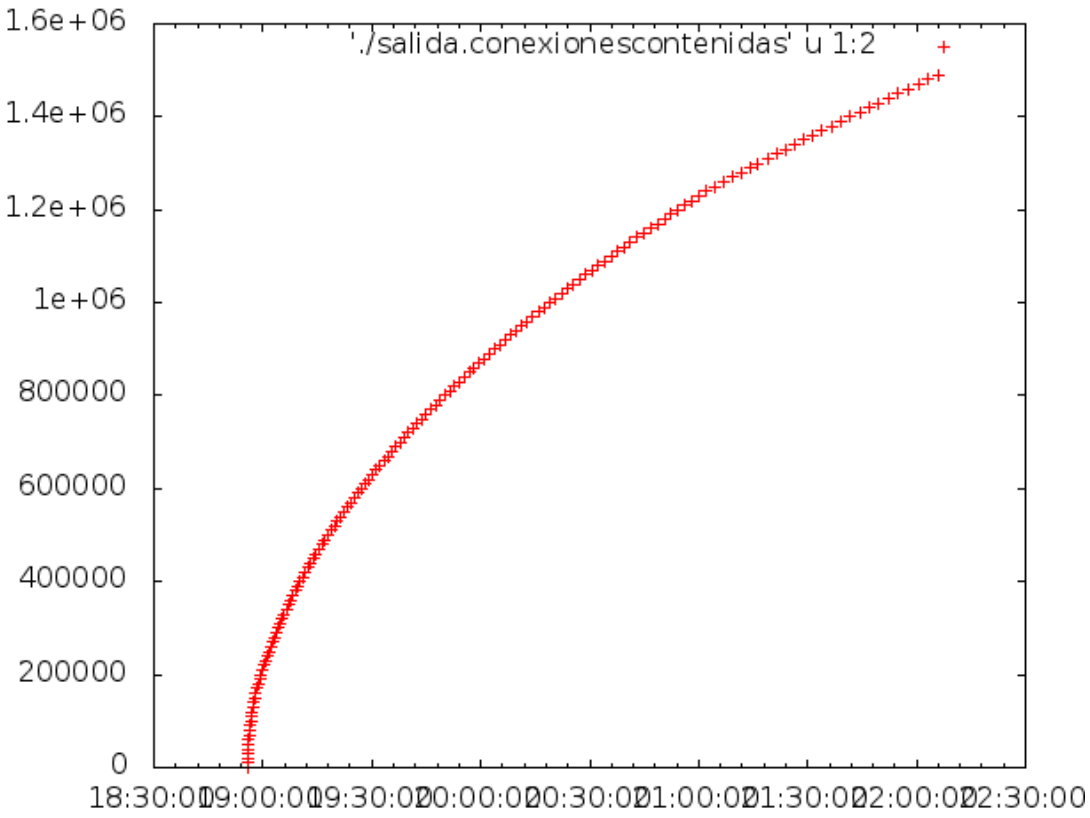

Figura 15-3.6: Tiempo práctico aumentando considerablemente el número de conexiones y parado el programa para hacerlo más visible

Una vez realizada la prueba con el fichero de 3GB obtenemos un resultado muy satisfactorio ya que, la ejecución es aproximadamente seis minutos, lo que nos permite hacer estudios acerca de la carga de las máquinas de la red en un tiempo razonable que es lo que buscábamos con este proyecto.

3.7 Conclusión

Por último vamos a analizar en conjunto los algoritmos utilizados.

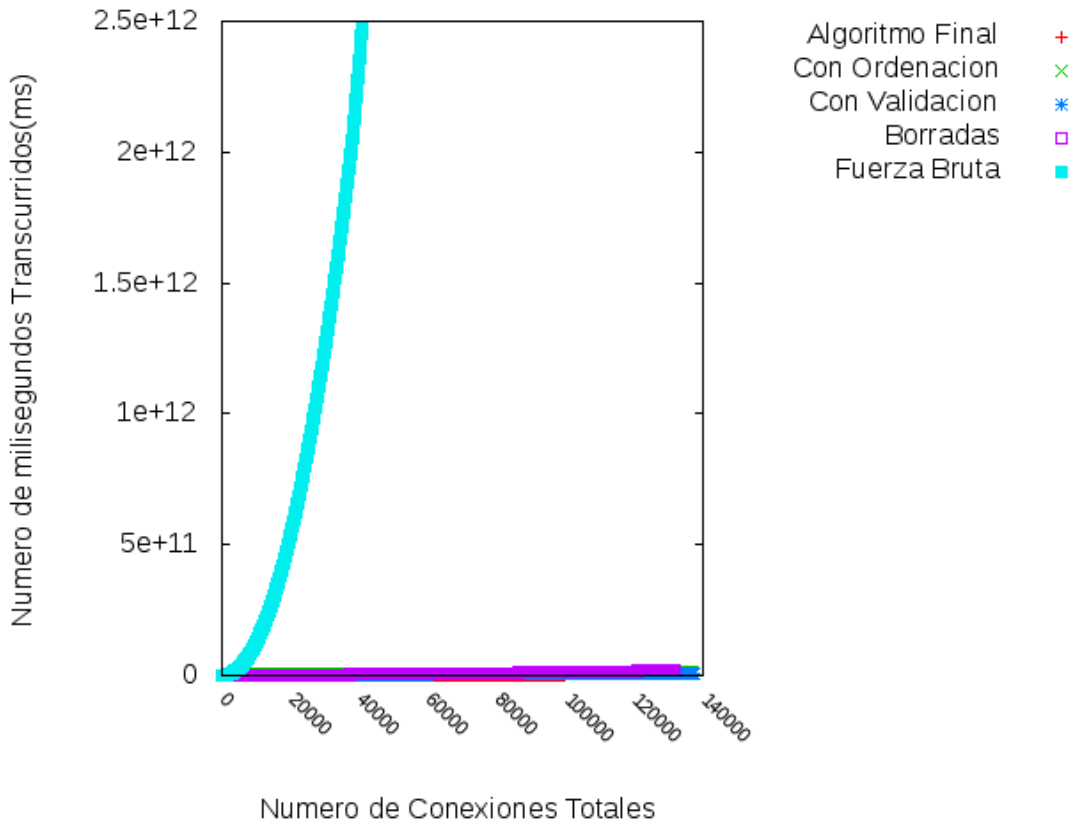

Figura 16-3.7: Tiempos prácticos en conjunto

Como podemos observar en la gráfica anterior, nos encontramos ante una mejora en el tiempo muy elevada entre el primer algoritmo y el resto, dado que los tiempos del primer algoritmo son muy elevados no tiene mucho sentido comparar los algoritmos en esta gráfica por lo tanto vamos a utilizar la figura 16-3.7, pero eliminando el tiempo del algoritmo de fuerza bruta.

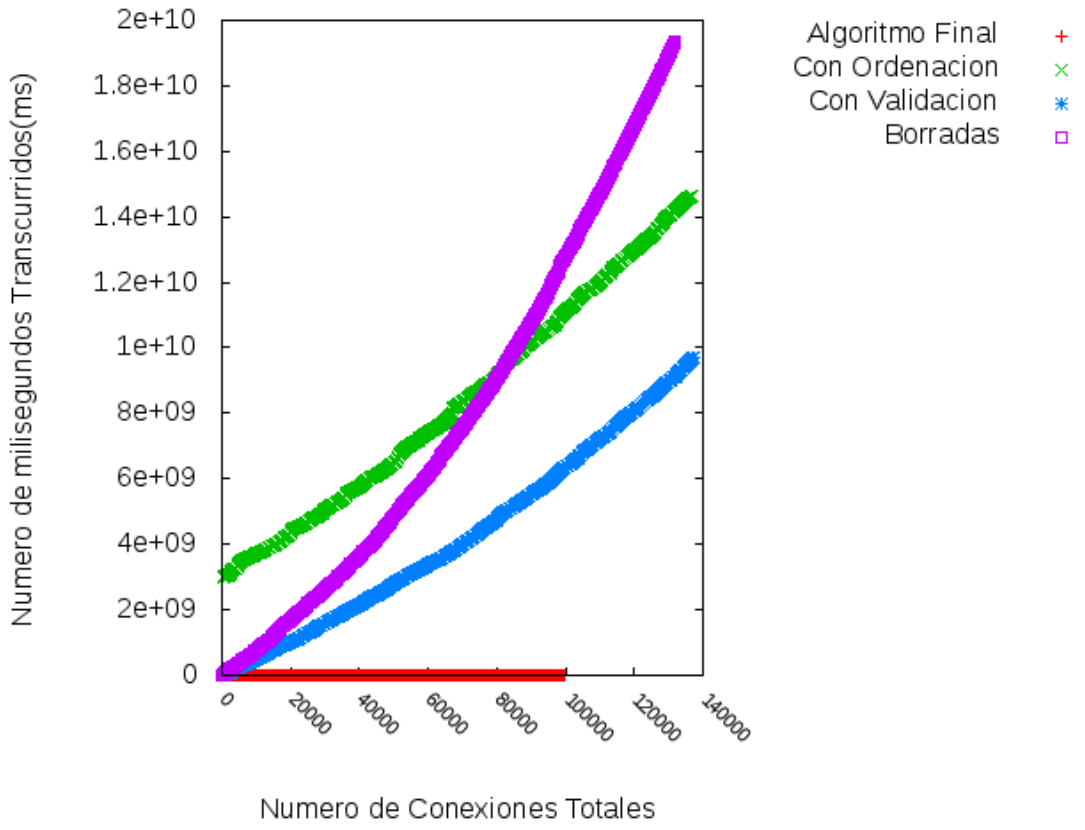

Figura 17-3.: Figura de tiempos Finales sin el algoritmo de fuerza bruta

Una vez hemos quitado ese tiempo tan elevado podemos apreciar lo que se comentaba anteriormente con respecto a los tiempos teóricos. Como podemos observar eliminando solo las conexiones tenemos el tiempo más elevado alcanzado, posteriormente como podemos observar tenemos la ordenación porque habíamos aumentado el tiempo en un factor N*log(N), por lo tanto hay tenemos el aumento a la hora de empezar con respecto al algoritmo de eliminación de conexiones no válidas.

Por último y cómo podemos observar nuestro algoritmo final no tiene un tiempo comparable con respecto a los anteriores.

4 Conclusiones y estudios futuros

4.1 Conclusiones teóricas y prácticas

Una vez desarrollado este trabajo llegamos a la conclusión de que el cálculo de conexiones contenidas se puede reducir de orden N^2 a N*log(N), lo que nos permite realizar estudios topológicos de la red analizada como por ejemplo, el número de bytes consumido por subredes de tipo A (aaaa:bbbb:cccc:dddd), así como los puertos más utilizados sobre los cuales se desarrollan las conexiones contenidas, con pequeños scripts realizados en AWK.

También se puede ver fácilmente que relajando la condición contenida del tiempo de fin con respecto a las dos conexiones, obtenemos las conexiones concurrentes que se dan sobre las diferentes máquinas de nuestra red, por lo tanto también se puede ver que el cálculo de conexiones concurrentes en el entramado de una red es de orden O(N*log(N)). Esto nos permite también realizar los mismos estudios que podemos realizar sobre las conexiones contenidas, sobre las conexiones concurrentes.

Por último esta cota es bastante importante debido a que es la primera cota que se da sobre este problema, ya que no hay nada publicado anteriormente acerca de este problema.

4.2 Estudios de futuro y posibles mejoras

Se puede mejorar este algoritmo de varias formas, la primera cambiando el algoritmo de ordenación, y la otra añadiendo multiproceso.

4.2.1 Cambio del algoritmo de ordenación

En estos momentos los algoritmos de ordenación que se utilizan son todos de orden N*log(N) aunque se están investigando algunos algoritmos que reducen esta cota, aunque no son siempre aplicables, sea el caso de RadixSort que tiene orden de O(N) pero si se utilizan cadenas, en cambio sí se utilizan números el orden de RadixSort no mejora con respecto a Quicksort o Mergesort.

Una mejora razonable sería el cambio del algoritmo Quicksort a Heapsort pero el coste de implementación de Heapsort, con respecto a Quicksort que ya está implementado en C es bastante elevado para el trabajo de fin de grado, porque hay que tener en cuenta las optimizaciones realizadas.

4.2.2 Añadir multiproceso en el bucle del cálculo de las conexiones contenidas

Para añadir multiproceso vamos a suponer primeramente que el número de conexiones a analizar es N y este es mayor que el número de procesos que vamos a utilizar que lo denotaremos por P.

Supongamos que tenemos nuestras conexiones y queremos partir este periodo de ejecución en varios trozos, lo que tendremos que hacer será lo siguiente:

Una vez tengamos las conexiones que queremos analizar, en este caso suponemos que ya hemos eliminado las conexiones que no son válidas y ya están ordenadas. Tenemos que repartir las conexiones entre los procesos que vamos a utilizar. A primera vista se nos puede plantear que lo sencillo es dividir N/P y coger su techo para calcular el número de conexiones, es decir, el primer proceso tendrá N/P conexiones el segundo 2*N/P… así hasta el proceso p-esimo que analizará las últimas conexiones. Pero esto no se puede hacer, ya que al cortar por un punto tienes que asegurarte que las conexiones anteriores a las que cortas están dentro de las analizadas.

Por consiguiente, tenemos que realizar lo siguiente:

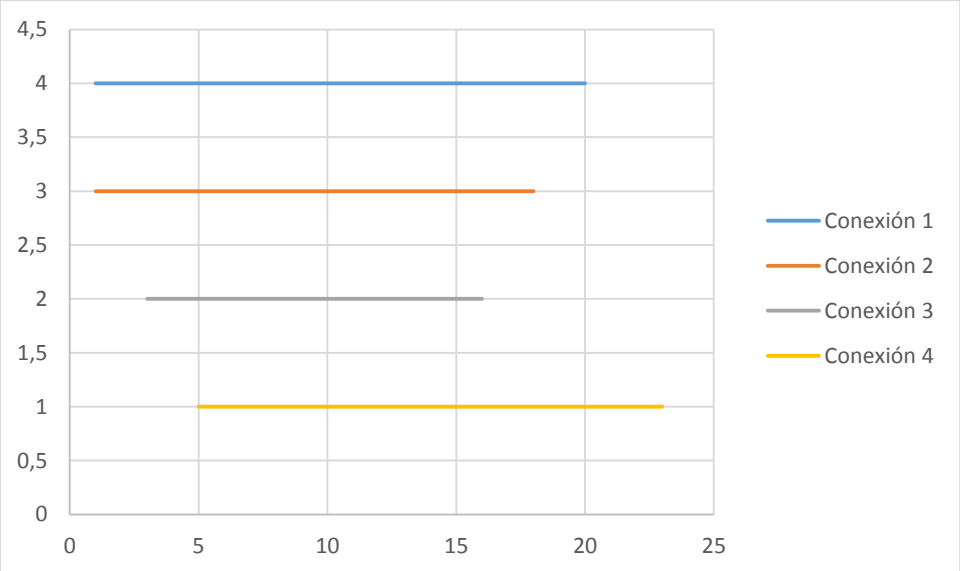

Figura 1-4.2.2: Conexiones ordenadas

Tras tener las conexiones como están pintadas en la figura anterior, seleccionamos el corte en el cual queremos dividir, es decir, el N/P. Ahora bien tenemos que tener en cuenta las anteriores.

Para ello lo que debemos hacer es marcar nuestra conexión supuestamente de inicio como primera conexión y posteriormente, tenemos que avanzar hacia atrás en el tiempo, hasta que el tiempo de inicio de nuestra “primera conexión” sea menor que el tiempo de fin de la conexión sobre la que estamos avanzando hacia atrás. Entonces tenemos ya nuestra conexión real de inicio y nuestra conexión final que vienen delimitada por el siguiente N/P.

El problema que se plantea, es que hay conexiones que estamos analizando más de una vez algunas conexiones, por lo tanto la mejora del algoritmo en esta parte no es como total 1/P si no algo menor.

Glosario de Terminos

Snifer: Programa que permite captar los paquetes que circulan por el entramado de la red, en este caso nuestro snifer aparte de eso nos da las conexiones realizadas.

Conexión: unión, no tiene por qué ser física, entre dos máquinas mediante la red.

Conexión Contenida: Es aquella conexión que su tiempo de fin es menor que el tiempo de fin de la conexión que lo contiene, y el tiempo de inicio es mayor que el tiempo de inicio de la conexión que la contiene

Conexión concurrente: Es aquella conexión que cumple que el tiempo de inicio de una es menor que el tiempo de fin que el de la conexión concurrente.

ANEXO

A Campos del fichero

En este anexo se encuentra reflejado los campos que tendrá el fichero que vamos a utilizar para el cálculo de conexiones contenidas:

1 srcIP Direccion IP del cliente de la conexión.

2 srcPort Puerto empleado por el cliente.

3 dstIP Direccion IP del servidor de la conexión.

4 dstPort Puerto empleado por el servidor.

5 firstPacketTime timestamp del primer paquete de la conexión

6 lastPacketTime timestamp del último paquete de la conexión

7 firstSYNSrcToDstTime timestamp del primer SYN de cliente a servidor

8 firstSYNDstToSrcTime timestamp del primer SYN del servidor al cliente

9 firstACKSrcToDstTime timestamp del primer ACK de cliente a servidor

10 firstACKDstToSrcTime timestamp del primer ACK de servidor a cliente

11 firstDataSrcToDstTime timestamp del primer paquete con datos de cliente a servidor

12 firstDataDstToSrcTime timestamp del primer paquete con datos de servidor a cliente

13 lastDataSrcToDstTime timestamp del último paquete con datos de cliente a servidor

14 lastDataDstToSrcTime timestamp del último paquete con datos de servidor a cliente

15 lastSrcToDstTime timestamp del último paquete de cliente a servidor

16 lastDstToSrcTime timestamp del último paquete de servidor a cliente

17 numberPacketsSrcToDst número de paquetes de cliente a servidor

18 numberPacketsDstToSrc número de paquetes de servidor a cliente

19 numberSYNSrcToDst número de SYNs de cliente a servidor

20 numberSYNDstToSrc número de SYNs de servidor a cliente

21 numberFINSrcToDst número de FINs de cliente a servidor

22 numberFINDstToSrc número de FINs de servidor a cliente

23 numberRSTSrcToDst número de RSTs de cliente a servidor

24 numberRSTDstToSrc número de RSTs de servidor a cliente

25 numberPacketsDataSrcToDst número de paquetes con datos de cliente a servidor

26 numberPacketsDataDstToSrc número de paquetes con datos de servidor a cliente

27 numberDupACKsSrcToDst número de ACKs duplicados de cliente a servidor (poco fiable)

28 numberDupACKsDstToSrc número de ACKs duplicados de servidor a cliente (poco fiable)

29 numberHolesSrcToDst número de huecos vistos en la secuencia de datos de cliente a servidor (nada fiable)

30 numberHolesDstToSrc idem

31 numberDisorderSrcToDst Número de segmentos con datos que llegan fuera de orden. Si llega un segmento pero faltan datos anteriores a él se considera que llega fuera de orden. Si luego llegan esos datos anteriores se cuentan también como fuera de orden

32 numberDisorderDstToSrc idem

33 bytesIPSrcToDst Número de bytes acumulados de paquetes IP de cliente a servidor sumando tamaños enteros de los paquetes IP

34 bytesIPDstToSrc idem

35 bytesPhySrcToDst Idem pero a nivel físico que en el caso de una Ethernet es contar el tamaño de la trama tal y como la da pcap (sin CRC)

36 bytesPhyDstToSrc idem

37 bytesTCPDataSrcToDst Suma de los bytes de datos de cliente a servidor que se han visto. Los datos en segmentos retransmitidos serán contados varias veces

38 bytesTCPDataDstToSrc idem

39 bytesIPDisorderSrcToDst bytes a nivel IP de los paquetes que se han considerado desordenados

40 bytesIPDisorderDstToSrc idem

41 bytesTCPDisorderSrcToDst bytes de datos TCP de paquetes desordenados

42 bytesTCPDisorderDstToSrc idem

43 ISNSrcToDst Número de secuencia inicial de cliente a servidor

44 ISNDstToSrc idem

45 minWindowSrcToDst Mínimo tamaño de ventana de control de flujo vista en anuncios de cliente a servidor

46 minWindowDstToSrc idem

47 maxWindowSrcToDst Máximo tamaño de ventana de control de flujo vista en anuncios de cliente a servidor

48 maxWindowDstToSrc idem

49 firstPacketFlags Flags del primer paquete que se ve, que si hay desorden puede no ser el primero de la conexión

50 firstPacketCounter Número de secuencia en la traza del paquete que es el primero de esta conexión (si se aplica un filtro se cuentan los paquetes una vez filtrados)

51 lastPacketCounter Idem pero último paquete de la traza

52 reasonStart La forma en que se inició el registro de la conexión en este procesado (0 normal, 1 se repite la cuatupla con un SYN con nuevo ISN)

53 reasonDump La razón por la que se terminó y volcó este registro de conexión (1 comienza otra conexión con misma cuatupla, 2 se ha excedido el tiempo máximo de inactividad, 3 termino la traza; máximo tiempo de inactividad por defecto es de 3600 segundos)

54 whenDecissionDump Timestamp del último paquete procesado cuando se decide volcar esta conexión

55 lastAckNumberSrcToDst Último número de ACK visto de cliente a servidor

56 lastAckNumberDstToSrc Último número de ACK visto de servidor a cliente

57 nextSeqNumberSrcToDst Proximo número de secuencia esperado de cliente a servidor porque es siguiente al mayor visto

58 nextSeqNumberDstToSrc Proximo número de secuencia esperado de servidor a cliente porque es siguiente al mayor visto

59 numberReTxSrcToDst Número de segmentos que contenían datos retransmitidos en el flujo de cliente a servidor

60 bytesTCPReTxSrcToDst Suma de longitudes de datos TCP de segmentos que tenían algún byte que era retransmitido (entiendo que serán retx todos los del segmento pero si hay unos retx y otros nuevos, por lo que sea, no se comprueba)

61 bytesPhyReTxSrcToDst Suma de longitudes a nivel por debajo de IP (físico/enlace) de todos los paquetes TCP que contenían números de secuencia que ya se habían visto antes (ojo, eso incluye retx de SYN y FIN)

62 huboHuecosSrcToDst Vale 1 si se detectó en algún momento que se produjeran huecos en la secuencia de cliente a servidor

63 numberReTxDstToSrc Número de segmentos que contenían datos retransmitidos en el flujo de servidor a cliente

64 bytesTCPReTxDstToSrc Suma de longitudes de datos TCP de segmentos que tenían algún byte que era retransmitido (entiendo que serán retx todos los del segmento pero si hay unos retx y otros nuevos, por lo que sea, no se comprueba)

65 bytesPhyReTxDstToSrc Suma de longitudes a nivel por debajo de IP (físico/enlace) de todos los paquetes TCP que contenían números de secuencia que ya se habían visto antes (ojo, eso incluye retx de SYN y FIN)

66 huboHuecosDstToSrc Vale 1 si se detectó en algún momento que se produjeran huecos en la secuencia de servidor a cliente

67 bytesPhyDisorderSrcToDst Suma de longitudes de tramas a nivel inferior a IP para segmentos desordenados en el flujo de cliente a servidor

68 bytesPhyDisorderDstToSrc Suma de longitudes de tramas a nivel inferior a IP para segmentos desordenados en el flujo de servidor a cliente

69 timeAckSrcToDstForLastDataDstToSrc timestamp del último ACK de cliente a servidor que confirma un segmento TCP con datos de servidor a cliente. Esto debería marcar una cota de cuando se han terminado de recibir todos los datos en el cliente. Puede ser un ACK anterior al FIN o puede ir con el FIN. Sera el primer ACK que confirme el número de secuencia más alto visto de servidor a cliente

70 timeAckDstToSrcForLastDataSrcToDst Análogo pero para el sentido

71 timeFirstFINSrcToDst timestamp del primer FIN de cliente a servidor

72 timeLastAckSrcToDst timestamp del último ACK de cliente a servidor

73 timeFirstFINDstToSrc timestamp del primer FIN de servidor a cliente

74 timeLastAckDstToSrc timestamp del último ACK de servidor a cliente

75 maxBlockedTimeSrcToDst Máximo tiempo que ha transcurrido entre dos segmentos TCP con datos de cliente a servidor sin que entre ellos circulara ningún paquete servidor a cliente

76 timeMaxBlockedTimeSrcToDst timestamp del paquete que cierra el mayor intervalo de tiempo entre dos segmentos TCP con datos de cliente a servidor sin que entre ellos circulara ningun paquete de servidor a cliente

77 maxBlockedTimeDstToSrc Análogo en sentido contrario

78 timeMaxBlockedTimeDstToSrc Análogo en sentido contrario

79 timeFirstSSLDataSrcToDst

80 timeFirstSSLDataDstToSrc

81 timeLastSSLDataSrcToDst

82 timeLastSSLDataDstToSrc

83 srcMACSrcToDst Direccion MAC origen de los paquetes de cliente a servidor en formato aa:bb:cc:dd:ee:ff

84 dstMACSrcToDst Direccion MAC destino de los paquetes de cliente a servidor en formato aa:bb:cc:dd:ee:ff

85 srcMACDstToSrc Direccion MAC origen de los paquetes de servidor a cliente en formato aa:bb:cc:dd:ee:ff

86 dstMACDstToSrc Direccion MAC destino de los paquetes de servidor a cliente en formato aa:bb:cc:dd:ee:ff

87 moreMACsSeen Vale 1 si se han visto más direcciones MAC que las 4 anteriores

88 numPktsFragmentos Número de paquetes IP que eran fragmentos. Solo cuenta el primer fragmento del segmento pues en el resto, al no estar la cabecera TCP no puede sacar los puertos y reconocer a que conexión corresponde

B Ejemplo de conexiones que aparecen en el fichero

Conexión 1:

7.2.204.240 63517 7.220.0.10 2065 1353477695.085357 1353477697.081568 1353477695.085357 1353477695.085922 -1 1353477695.085922 -1 -1 -1 -1 1353477697.081568 1353477695.085922 3 1 2 1 0 0 1 0 0 0 0 0 0 0 0 0 92 24 200 60 0 0 0 0 0 0 127349340 4275710485 0 4128 4128 4128 2 628608 646906 0 1 1353477697.082166 -1 127349341 127349341 4275710486 0 0 0 0 0 0 0 0 0 0 -1 1353477695.085922 -1 -1 -1 1353477695.085922 0.000000 -1 0.000000 0.000000 -1 -1 -1 -1 00:1c:57:13:94:1b 00:0e:39:db:48:00 00:0e:39:db:48:00 00:1c:57:13:94:1b 1 0

Conexión 2:

7.2.204.240 23360 7.220.0.11 2065 1353477695.082771 1353477697.081502 1353477695.082771 1353477695.083443 -1 1353477695.083443 -1 -1 -1 -1 1353477697.081502 1353477695.083443 3 1 2 1 0 0 1 0 0 0 0 0 0 0 0 0 92 24 200 60 0 0 0 0 0 0 606485731 517647617 0 4128 4128 4128 2 628568 646907 0 1 1353477697.082197 -1 606485732 606485732 517647618 0 0 0 0 0 0 0 0 0 0 -1 1353477695.083443 -1 -1 -1 1353477695.083443 0.000000 -1 0.000000 0.000000 -1 -1 -1 -1 00:1c:57:13:94:1b 00:0e:39:db:48:00 00:0e:39:aa:e4:00 00:1c:57:13:94:1b 1 0

Conexion3:

7.2.204.240 22282 7.220.0.12 2065 1353477695.083865 1353477697.081507 1353477695.083865 1353477695.084829 -1 1353477695.084829 -1 -1 -1 -1 1353477697.081507 1353477695.084829 3 1 2 1 0 0 1 0 0 0 0 0 0 0 0 0 92 24 200 60 0 0 0 0 0 0 3483308291 3088520685 0 4128 4128 4128 2 628585 646910 0 1

1353477697.082374 -1 3483308292 3483308292 3088520686 0 0 0 0 0 0 0 0 0 0 -1 1353477695.084829 -1 -1 -1 1353477695.084829 0.000000 -1 0.000000 0.000000 -1 -1 -1 -1 00:1c:57:13:94:1b 00:0e:39:db:48:00 00:0e:39:aa:e4:00 00:1c:57:13:94:1b 1 0

Conexión 4:

7.2.204.240 23360 7.220.0.11 2065 1353477697.082197 1353477701.080507 1353477701.080507 1353477697.082197 -1 1353477697.082197 -1 -1 -1 -1 1353477701.080507 1353477697.082197 2 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 44 24 120 60 0 0 0 0 0 0 606485731 139179120 0 4128 4128 4128 12 0 676982 1 1 1353477701.081149 -1 606485732 606485732 139179121 0 0 0 0 0 0 0 0 0 0 -1 -1 -1 -1 -1 1353477697.082197 0.000000 -1 0.000000 0.000000 -1 -1 -1 -1 00:1c:57:13:94:1b 00:0e:39:db:48:00 00:0e:39:db:48:00 00:1c:57:13:94:1b 0 0

...

C Ejemplo de salida del programa final

Ejemplo de salida del programa:

156.1.0.10 8241 150.250.40.65 5021 0 1279 1353477667.917628 1353477668.228179

156.1.0.10 8250 150.250.40.65 21 0 1279 1353477667.917628 1353477668.228179

156.1.0.10 8258 150.250.40.65 5021 0 1589 1353477747.369576 1353477747.648406

Campos que muestra:

1 Ip Destino

2 Puerto Destino

3 Ip Origen

4 Puerto Origen

5 Bytes Consumidos por el Cliente

6 Bytes Consumidos por el Servidor

7 Tiempo de Inicio de la conexión contenida

8 Tiempo de Fin de la conexión contenida